68' MICRO JOURNAL

Australia A $ 4.75 | New Zealand NZ $ 6.50
Singapore S $ 9.45 | Hong Kong H $23.50
Malaysia M $ 9.45 | Sweden 30:-SEK

$2.95 USA

Motorola VME-MACINTOSH-S 50 & Other 68XXX Systems

6809 68008 68000 68010 68020 68030

OS-9

The Magazine for Motorola CPU Devices
For Over a Decade!
A User Contributed Journal

FLEX
SK*DOS

This Issue:

And Lots More!

VOLUME IX ISSUE V ● Devoted to the 68XXX User ● May 1987

"Small Computers Doing Big Things"

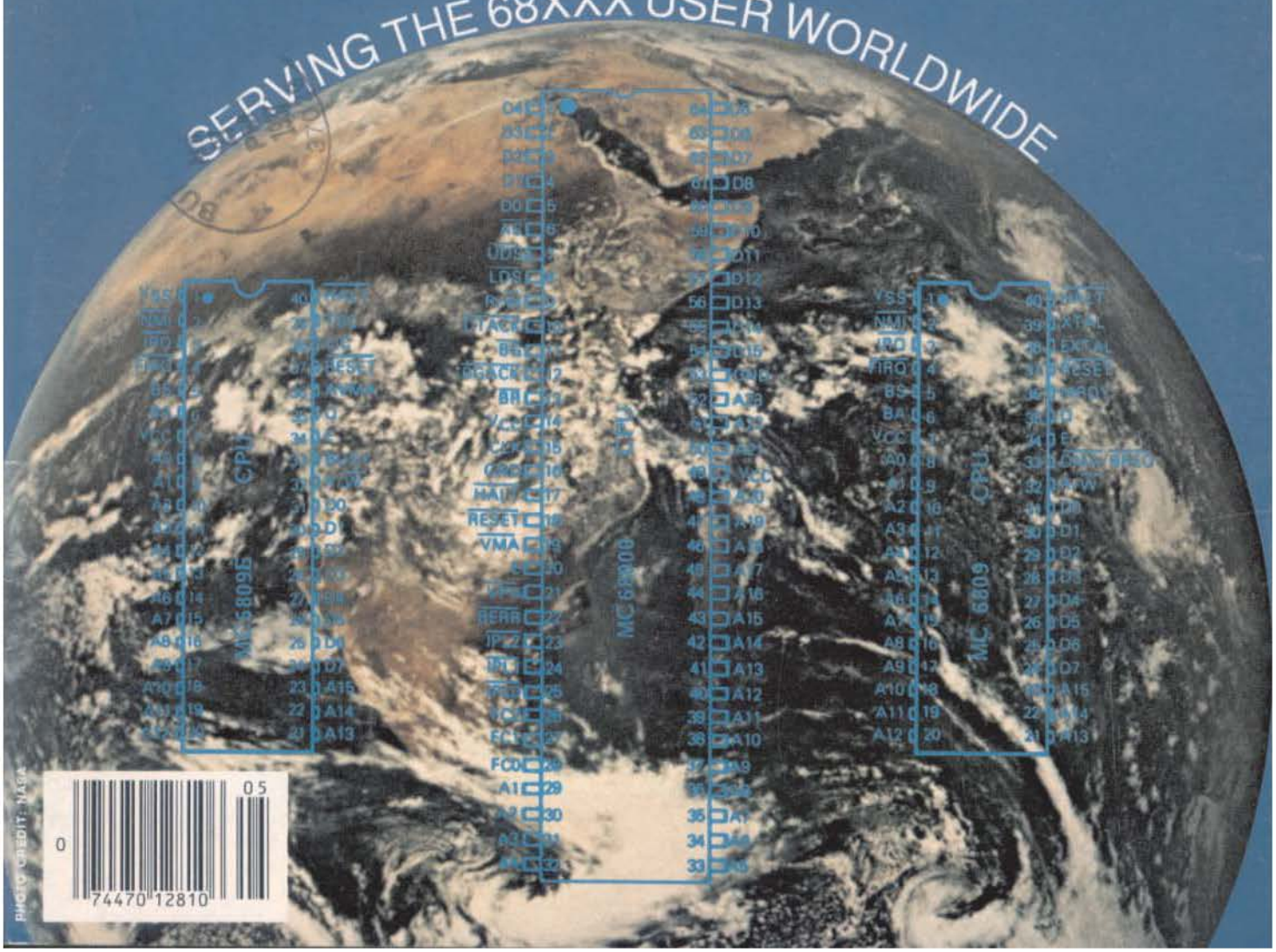
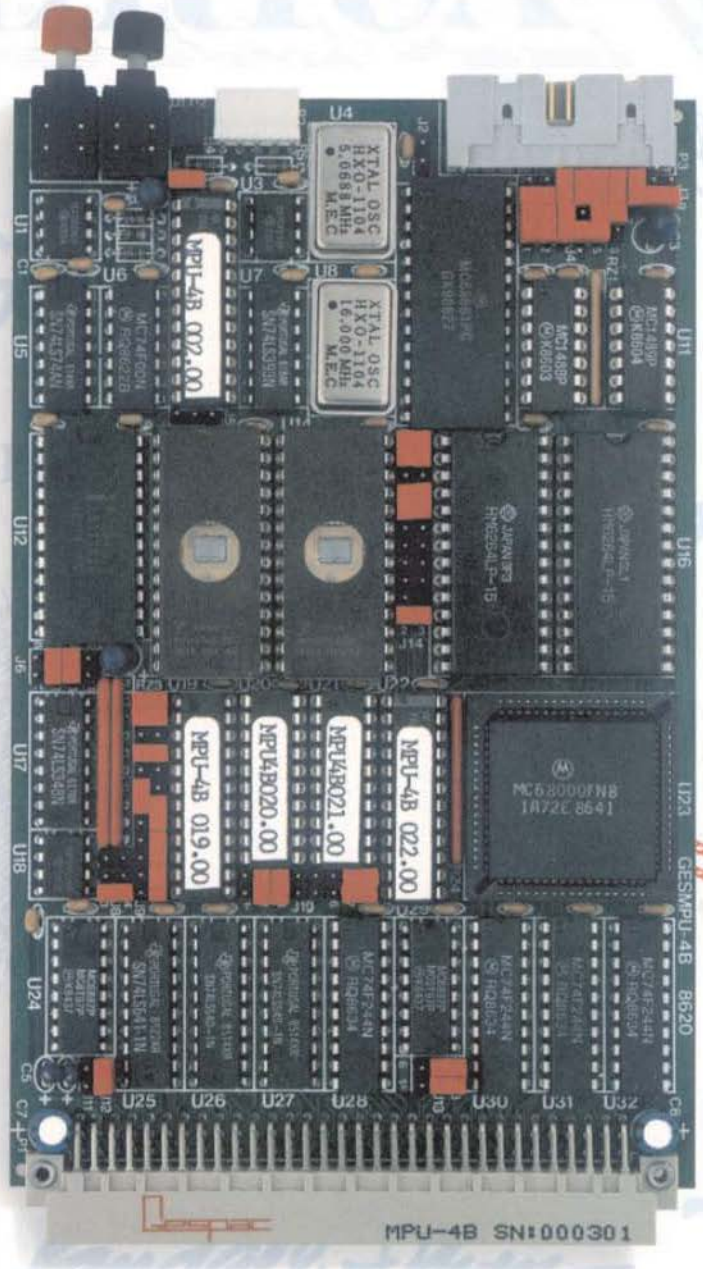


PHOTO CREDIT: NASA

0 74470 12810 05

GESPAC Gives You 68000 Performance at 8-bit Prices.



Actual Size

Introducing another Price / Performance breakthrough from GESPAC: A complete 68000 CPU module for $395 unit price. The GESMPU-48.

Wherever you thought of using an 8-bit microprocessor to do a job, you can now use the 16/32-bit GESMPU-48 instead. It will do the job, better, faster, and best-of-all for the same money.

On a single height Eurocard, we have packed an 8 MHz 68000 microprocessor (16 MHz optional), four sockets for up to 64 Kilobytes of zero-wait-stotes CMOS RAM and up to 128 Kilobytes of EPROM, one RS-232 serial port, and three 16-bit timers.

The GESMPU-48 is fully expandable through the standard G-64 bus, to accommodate up to 16 Megabytes of external memory. You can add any of more thon 300 I/O modules available from GESPAC and a growing numbers of independent G-64 bus vendors.

To make your programming tasks easier, GESPAC supports the GESMPU-48 with the OS-9® and PDOS® real-time, multi-tasking operating systems and most popular high level languages and software development tools.

$395.00
Single Quantity

$316 @ 100 pieces

If you too like the ideo of getting more for less, contact us today to receive information on the GESMPU-4B and the G-64 bus concept from GESPAC—the leader in single Eurocard microcomputer products worldwide.

Call Toll Free 1-800-4-GESPAC
or Call (602) 962-5559.

Gespac

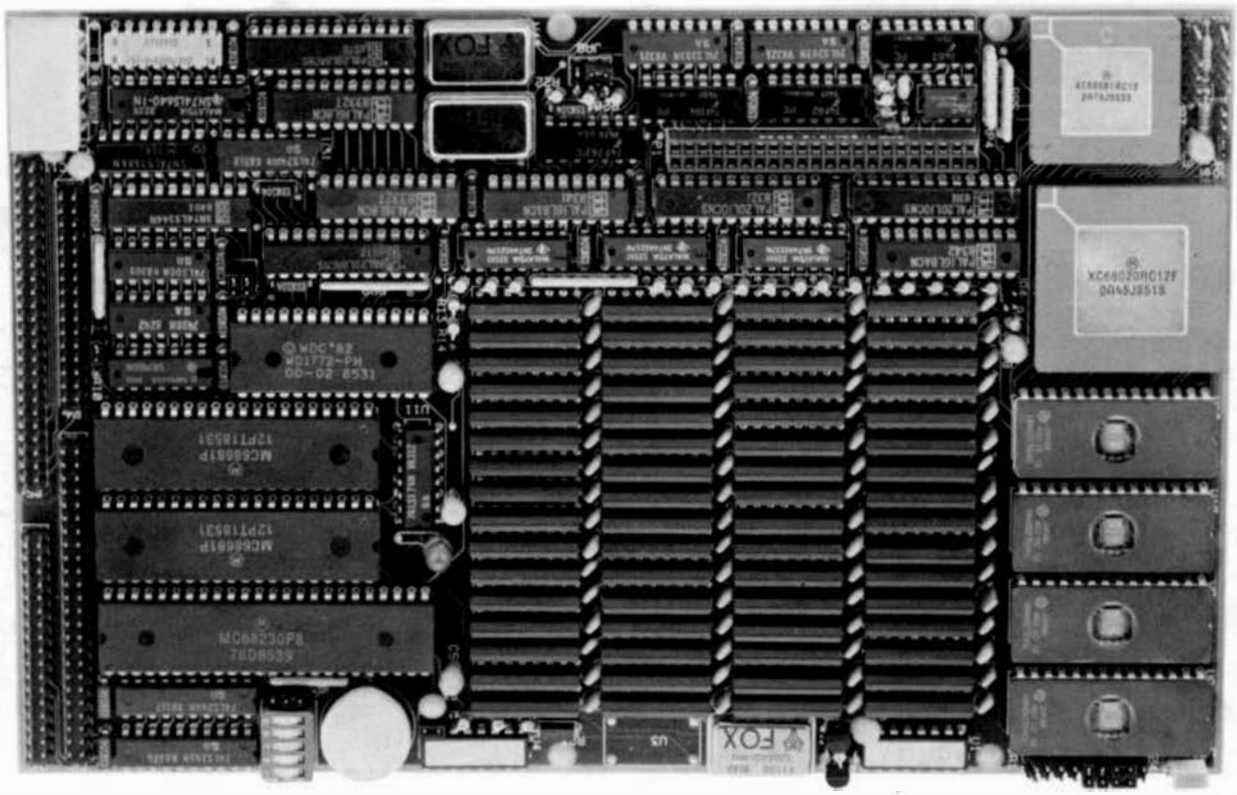
IN USA - CANADA
50A West Hoover Ave.
Mesa, Arizona 85202
Tel. (602) 962-5559
Telex 386575

INTERNATIONAL
3, chemin des Aulx
CH-1228 Geneva
Tel. (022) 713400
Telex 429989

GMX™ Micro-20
68020 SINGLE—BOARD COMPUTER

Mainframe CPU Performance on a 5.75" x 8.8" Board
(benchmark results available on request)

$2565.00 12.5 MHz Version
Quantity Discounts Available



Features

- 32-Bit MC68020 Processor (12.5, 16.67, or 20MHZ)
- MC68881 Floating-point coprocessor (optional)
- 2 Megabytes of 32-bit wide, high-speed RAM
- 4 RS-232 Serial I/O Ports (expandable to 36)
- 8-bit Parallel I/O Port ('Centronics' compatible)
- Time-of-Day Clock w/ battery backup
- 16-bit I/O Expansion Bus
- Up to 256 Kbytes of 32-bit wide EPROM
- Floppy Disk Controller for two 5¼" drives
- SASI Intelligent Peripheral Interface (SCSI subset)
- Mounts directly on a 5¼" Disk Drive
- Optional Boards Include Arcnet, Prototyping, I/O Bus adapter, 60 line Parallel I/O, RS-422/485

Software

Included:
- GMX Version of Motorola's 020Bug Debugger with up/download, breakpoint, trace, single-step, and assembler/disassembler capabilities
- Comprehensive Hardware Diagnostics

Optional:

UNIX™ like Multi-user/Multi-tasking Disk Operating Systems
- OS-9/68000™ (Real-time and PROMable)
- UniFLEX™

Programming Languages and Application Software
- BASIC, C, PASCAL, ABSOFT FORTRAN, COBOL and ASSEMBLER
- Spreadsheet, Data Base Management, and Word Processing

COMPLETE EVALUATION SYSTEMS AVAILABLE

GMX inc. 1337 W. 37th Place Chicago, IL 60609
(312) 927-5510 • TWX 910-221-4055
State-of-the-Art Computers Since 1975

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.
"Over a Decade of Service"
"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN. and additional entries. Posunaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year **$24.50** USA, Canada & Mexico **$34.00** a year. Others add **$12.00** a year surface, **$48.00** a year Airmail, USA funds. 2 years **$42.50**, 3 years **$64.50** plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

Mustang-020 Mustang-08 Benchmarks

```
                                                    32 bit      Register
                                                    Integer     Long

IBM AT 7300 Xenix Sys 3                               9.7
AT&T 7300 UNIX PC 6.2.0.10                            7.2         4.3
DEC VAX 11/780 UNIX Berkley 4.2                       3.6         3.2
DEC VAX 11/750     "       "                          5.1         3.2
68000 OS-9 6BK 8 Mhz                                 10.0         9.0
68000 OS-9 6BK 10 Mhz                                 6.5         4.0
MUSTANG-08 68008 OS-9 6BK 10 Mhz                      9.8         6.3
MUSTANG-020 68020 OS-9 6BK 16 Mhz                     2.3         0.88
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz              1.8         1.25

Main()
    {
            register long i;
            for (i=0; i < 999999; ++i);
    }
Estimated MIPS - MUSTANG-020 ..... 4.5 MIPS,
Burst to 8 - 10 MIPS: Motorola Specs
```

OS-9

OS-9 Professional Ver *Includes C Compiler	$350.00
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Fortran 77	750.00
Microware Pascal	500.00
Omegasoft Pascal	900.00
Style-Graph	495.00
Style-Spell	195.00
Style-Merge	175.00
Style-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	269.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PostCompiler	500.00
C Compiler	350.00
CDBUG	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Sculptor+ (see below)	995.00
Standard MUSTANG-020™ shipped 12.5 Mhz.	
Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	575.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232 requires 1 or 2 Adapter Cards below	335.00
RS232 Adapter Each card supports 4 additional ser. ports (total of 36 serial ports supported)	165.00
60 line Parallel I/O card Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.	398.00
Prototype Board area for both dip and PGA devices & pre-wired memory area up to 512K DRAM.	75.00
SBC-AN Interface between the system and ARCNET coax/fiber token passing LAN, fiber optics optional - call.	475.00
LAN software drivers	120.00
Expansion for Motorola I/O Channel Modules	$195.00

Special for complete MUSTANG-020™ system buyers - Sculptor+ $695.00. SAVE $300.00

Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

THE PRO!

Only the "PRO" version of OS-9 supported!

This is *HEAVY DUTY* Country!

UPGRADES
Write or Call for Professional OS-9 "Full Bore" Upgrade Kit

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information.

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
buffered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

Mustang Hi-Speed Systems Only from Data-Comp Div.
68008-68030

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

These hi-speed 68020 systems are presently working at *NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers*, worldwide, where speed, math crunching and multi-user, multi-tasking *UNIX C level V* compatibility and low cost is a must!

MUSTANG-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy	DS/DD	$269.95
Floppy cable		$39.95
OS-9 68K Professional Ver. * Includes C Compiler ($500.00)		$350.00
Winchester cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk controller		$395.00
Shipping USA UPS		$20.00
Total:	Save $1000.00	$5,299.80
complete sytem $4,299.80		
UniFLEX	Less	$100.00
MC68881 flp math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00

Note all 68881 chips work with 20 Mhz Sys

NOTE: Only Professional OS-9 now available (68020 Version) Includes ($500.00) C Compiler 68020 & 68881 supported

complete
25 Mbyte HD System
$4299.80
85 Mbyte HD System
$5748.80

Data-Comp Division
A Decade of Quality Service
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9 68XXX Systems

PAT FROM S. E. MEDIA -- A FULL FEATURED **SCREEN ORIENTED TEXT EDITOR** with all the best of PIE. For those who swore by and loved PIE, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included.** Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

The C Programmers Reference Source. Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

This chapter discusses command-line processing and the use and misuse of the C comma operator.

COMMAND - LINE PROCESSING

Command line arguments provide a convenient and powerful means of specifying certain actions of programs being executed. Operating systems such as UNIX, Uniflex, and OS/9-68K which support wildcard file names on the command line by passing a pseudo-command line to the program in which the file names have been expanded into file name lists. Operating systems which support environment variables evaluate these values when they are coded on the command line.

C programs may gain access to command line arguments with the argc and argv arguments of the main function. The following program outputs its list of command line arguments:

```
#include <stdio.h>

main(argc, argv)
int argc;
char *argv[];
{
    int i;

    for (i = 0; i < argc; ++i)
        printf("%d %s\n", i, argv[i]);
}
```

The getopt function, which is available in many standard C libraries, provides a standard means of scanning arguments on the command line. It is defined as follows:

```
extern char *optarg;
extern int optind, opterr;

int getopt(argc, argv, optstring);
int argc;
char **argv, *optstring;
```

The getopt function returns the next option letter which matches one of the letters in optstring. Optstring contains a list of known options; if a colon appears in optstring, the preceding letter is assumed to be an option which takes an argument. Optarg points to the start of the option argument, if any. Optind is the index of the next argument to be processed. When all options have been processed, getopt returns EOF. If an error is encountered, getopt returns a question mark and, if opterr is zero, outputs an error message to stderr.

Given a standard means such as the getopt function with which to process command line arguments and the ability of a C programmer to write custom means of scanning the command line, there remain decisions to be made concerning the level at which to issue prompts if the command line does not conform to expectations.

The most terse and unfriendly response is for the program to bomb, return without output, or issue a message such as "ERROR" and then to return. Unfortunately, many C programs take this approach. Many other C programs ignore invalid command line arguments and process the valid ones without other response.

A somewhat better response by a C program to invalid arguments on the command line is to provide a one-line prompt providing the format and order of arguments expected, without explanation. For example, the following line might represent this type of prompt

usage: s1bin infile outfile [lowaddr [highaddr [offset [splitfile]]]]

An even better response by a C program to invalid arguments is to provide a help screen providing much more complete information about the use of the program, including all or most command line arguments and options. Consider the following sample output:

```
usage: s1bin infile outfile [lowaddr
[highaddr [offset [splitfile]]]]
creates binary file from S1 records

 where: infile = input file name
outfile = output file name
lowaddr = hex low address
highaddr = hex high address
offset = hex offset value
splitfile = split file name
(if specified, odd addresses
are placed in splitfile and even
addresses are placed in outfile)
```

With help of this nature, the user should seldom be forced to review a program listing or manual to determine how to use a program. OS-9/68000 has a suggested standard of a command line argument "-?" used to obtain help about a program. This is an excellent idea, and considerably increases the level of usability of a system which consistently uses such standards. When combined with other standards, such as options starting with hyphens and (not) being case-sensitive, the level of usability of a system is again materially improved.

THE C COMMA OPERATOR

In the C language, the comma symbol is used for several purposes. In declarations, in initializers, and in macro and function headers and calls, commas separate related items. In expressions, otherwise unattached commas are considered arithmetic operators causing left-to-right sequential evaluation of their sub-expressions. The result of a comma operator is the type and value of the expression following the comma. The comma operator, like much of the rest of the C language, may be correctly used, to great advantage or incorrectly used, to great distress.

The following statement:

```
if (i < 5)
        ++i, ++k;
```

is equivalent to the following statements:

```
if (i < 5)
{
    ++i;
    ++k;
}
```

providing a means of coding certain statements in a shorthand fashion.

Similarly, the following statement:

```
if (i < 5)
    j = ++i, ++k, ++l;
```

is equivalent to the following statements:

```
if (i < 5)
{
    ++i;
    ++k;
    j = ++l;
}
```

which is legal, assuming that j and l are assignment-compatible.

The most convenient uses of the comma operator, other than the shorthand uses shown

above, are probably in the return and for statements.

The return statement has the following formats:

return

and

return expression

Folowing the first example above, the following statement:

```
if (i < 5)
    return (++i, ++k);
```

equivalent to the following statements:

```
if (i < 5)
{
    ++i;
    return (++k);
}
```

from which the type of final value and type of the variable k is returned, although the value of the variable i is incremented before the value of k is incremented.

The for statement has the following format:

for (expression1; expression2; expression3)

which is equivalent to the following statements:

```
{
    expression1;
    while (expression2)
    {
  :
  :
  expression3;
    }
}
```

Frequently, multiple initializations are required before the controlled statement is executed the first time or multiple changes are required for the next iteration of the controlled statement. The comma operator is often used to make the expression of these requirements much simpler.

For example, consider the following for statement:

```
for (i = x, j = i + 1; i < k; ++i, ++j)
{
    :
    :
}
```

in which the first and third expressions contain comma operators. Because of the left-to-right order of evaluation of expressions connected by comma operators, expressions dependent upon the results of previous expressions may be safely coded using comma operators as separators.

One of the most common misuse of comma operators is to assume that function arguments are evaluated left-to-right because the parameter list looks as if it were separated by comma operators rather than comma separators. The C language would have been improved if K&R or the new ANSI standard C had specified left-to-right evaluation of parameter lists, for consistency with comma operators.

Another misuse of comma operators is in multiple subscripts. Rather than var[i][j], it is easy to code var[i,j], especially for BASIC, COBOL, or FORTRAN programmers learning the C language. Many C compilers will not generate error messages in this case, leaving the inexperienced C programmer to find a subtle problem.

Still another misuse of comma operators is in the misplacement of parentheses in complex expressons containing commas and parentheses. For example, the following statement will compile properly on most C compilers, but will not execute correctly:

```
if (!(i = read(fd, buffer),
        sizeof (struct str)))
{
    :
    :
}
```

because the statement should have been coded as follows:

```
if (!(i = read(fd, buffer,
    sizeof (struct str))))
{
    :
    :
}
```

C PROBLEM

A BASIC program is being translated into the C language. The following BASIC declaration and statement are encountered:

```
DIM A(10,10)
A(10,10) = A(10,10) + 1
```

and are translated into the following C declaration and statement:

```
int a[10,10];
a[10,10] = a[10,10] + 1;
```

There are mistakes in this translation, decisions to be made, and optimizing opportunities not applied here.

One of the decisions which must be made concerns the use of base zero or base one in array declarations. BASIC processors differ in assuming either base zero or base one for arrays, but few BASIC programs use zero subscripts, even if the processor allows them. The use of the zero subscript in C programs is automatic, but may waste significant amounts of storage in multi-dimensional arrays. However, the translation is more complicated since every reference to a base one subscript translated to a base one subscript requires a subtraction.

Another decision concerns the type of each variable. Most BASIC processors assume that all variables are have type of floating point by default. In many cases, the C program may be easily optimized by using types of char, short int, int, or long, rather than types of float or double, where appropriate.

Two of the mistakes lie in the translation of A(10,10) used as a BASIC declaration or expression primary to a[10,10]. The declaration should be either a[11][11] or a[10][10], depending upon whether zero subscripts are assumed in the BASIC program or not. If the declaration is a[10][10], all subscripts must be decremented by one.

One of the optimization opportunities involves the statement incrementing A[10,10] by one. Most C compilers will not optimize this statement written as it is, but will correctly process it. Rewriting it as a pre-incrementation allows most C compilers to optimize it.

```
int a[11][11];
++a[10][10];
```

This problem was included because it illustrates several of the more common problems encountered by BASIC programmers when they begin to code C programs. In the case above, many C compilers will silently compile the code, treating [10,10] as if it were coded [10] (remember the hazards of the comma operator?).

Another common problem encountered by beginning C programmers is the use of the assignment operator '=' when the boolean equal operator '==' was intended. For the next C problem, consider what other features of the C language might present problems or might escape the scrutiny of those programmers already familiar with other languages.

EXAMPLE C PROGRAM

Following is this month's example C program; it illustrates one of the simpler forms of command line processing, using the getopt function described above. It provides only the simplest form of prompt, probably because the format is so simple and the user is assumed to have the listing available for reference.

```
/*
   The Tower of Hanoi problem consists of three
posts and a set of discs of increasing sizes.
The initial configuration places all the discson
one post arranged with small discson large ones.
The goal is to transfer all the discs to another
post.  There are two restrictions: only one disc
can be moved at any time, and it is illegal to
put a larger disc on a smaller.

   This implementation depends on two
   libraries: The curses screen handling
   package is used to make the display.
   The getopt function is used to handle
   the command line options.
   The options control the size of the
   problem and the display style.
```

```
Number of discs:
       The default number of discs is
       NODISCS, defined below.  The -n
       option can change this up to a
       maximum of MAXOISCS; the maximum
       is based on a terminal screen
       with 80 columns.
         Display Speed:
       By default, a smooth animation
       algorithm is used.  The -f (fast)
       option makes hanoi less smooth,
       displaying discs at key points in
       their transfers.  This is a good
       choice for slow speed terminals
       because there is still a motion
       illusion.  The -t (teleport)
       option shows discs only after
       they have been moved.
     */

     #include <stdio.h>
     #include <ctype.h>
     #include <curses.h>

     #define LEFT 0
     #define MIDDLE 1
     #define RIGHT 2
     #define MAXDISCS 13
     #define TOP (MAXDISCS+5)
     #define BLANK ' '
     #define DISC '='
     #define NOOISCS 7

     #define SMOOTH 0
     #define FAST 1
     #define TELEPORT 2

     /* Display: SMOOTH, FAST, or TELEPORT */
     int Display = SMOOTH;
     /* Number of discs, up to MAXDISCS */
     int Nodiscs = NODISCS;
     /* number of discs on each post */
     int count[3];
     /* column number of each post */
     int col[3];

     main (argc, argv)
     char **argv;
     {
         initial (argc, argv);
        hanoi (Nodiscs, LEFT, MIDDLE, RIGHT);
         sleep (Nodiscs);
        hanoi (Nodiscs, RIGHT, MIDDLE, LEFT);
         finish ("Done!");
     }

finish (s)
char *s;
{
    move (LINES-2, 0);
    refresh ();
    endwin ();
    puts (s);
    puts("\033\\");
    exit (0);
}

initial (argc, argv)
char **argv;
{
    int errflg = 0; /* option errors */
    int C;    /* option flag */
    extern char *optarg;

    puts("\033]");

    while ((C = getopt (argc,
  argv, "tfn:")) != EOF)
    {
  switch (C)
  {
  case 'f':
Display = FAST;
break;
  case 't':
Display = TELEPORT;
break;
  case 'n':
Nodiscs = atoi (optarg);
if (Nodiscs < 1)
     Nodiscs = NODISCS;
if (Nodiscs > MAXDISCS)
     Nodiscs = MAXDISCS;
break;
  default:
errflg++;
  }
    }
    if (errflg)
    {
  fprintf (stderr,
"USAGE: %s ", argv[0]);
  fprintf (stderr,
"[-stf] [-n discs]\n");
  exit (1);
    }
    col[LEFT] = Nodiscs;
    col[MIDDLE] = 3*Nodiscs;
    col[RIGHT] = 5*Nodiscs;
    initscr ();
    noecho ();
    display (Nodiscs);
}
```

```
display (n)
{
    clear ();
    standout ();
    mvprintw (0, 0, " Tower of Hanoi ");
    standend ();
    while (n > 0)
  mkdisc (n--, LEFT);
}

hanoi (n, start, inter, finish)
{
    if (n > 0)
    {
  hanoi (n-1, start, finish, inter);
  movedisc (n, start, finish);
  hanoi (n-1, inter, start, finish);
    }
}

movedisc (n, start, finish)
{
    rmdisc (n, start);
    showdisc (n, start, finish);
    mkdisc (n, finish);
}

showdisc (n, start, finish)
{
    int dir;
    int y;
    int x;

    if (Display == TELEPORT)
  return;
    dir = (col[start] < col[finish]) ?
  1 : -1;
   for (y = count[start]; y < Nodiscs-1;
  y++)
    {
  plotdisc (n, y, col[start],
BLANK);
  if (Display == SMOOTH)
refresh ();
  plotdisc (n, y+1, col[start],
DISC);
  if (Display == SMOOTH)
refresh ();
    }
    refresh ();
  for (x = col[start]; x != col[finish];
    x += dir)
    {
  plotdisc (n, y, x, BLANK);
  plotdisc (n, y, x+dir, DISC);
  if (Display == SMOOTH)
refresh ();
    }
    refresh ();
    while (y > count[finish])
    {
  plotdisc (n, y--, col[finish],
BLANK);
  if (Display == SMOOTH)
refresh ();
  plotdisc (n, y, col[finish],
DISC);
  if (Display == SMOOTH)
refresh ();
    }
    refresh ();
}

mkdisc (n, pile)
{
    plotdisc (n, count[pile], col[pile],
  DISC);
    count[pile]++;
    refresh ();
}

rmdisc (n, pile)
{
    count[pile]--;
    plotdisc (n, count[pile], col[pile],
  BLANK);
    refresh ();
}

plotdisc (n, y, x, c)
{
    int i;

    move (TOP-y, x-n);
    for (i = 1; i <= 2*n; i++)
  addch (c);
}
```

EOF

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!
6809 - 68020

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL
60139

LEAVING FOOTPRINTS

One of my thoughts this month is about documentation. Have you ever gone back to some source code you wrote some time back, only to find you can't decipher what your were doing? Or better yet, ever try reading someone else's code? Especially when they believe that documenting is for the other guy! I remember one source code that had only one comment. "GOOD LUCK!" That statement probably covers it all.

I am not innocent! I usually try to document my code, at least to the point of telling what a portion of code does. I am not one to do every line, but rather to comment on blocks of code. But beyond this, I have been a little sloppy. Occasionally, I have paid for it.

A few years back I had the job of doing a rewrite of some code written in Basic. Now the original author had not done a bad job writing the program, but there was little documentation. Actually, it was not all his fault. He had developed the program on a 32K machine. The source code was being run in an interpretive mode, which meant all the source occupied memory. Every REM statement used valuable memory. His program was long. He could not afford to waste precious bytes on much documentation. Eventually, the 32K computer and Basic gave way to a PDP-11, running Fortran. This meant a major rewrite of the code.

To this end
I would like to propose
some standards.

My standards
are not
chiseled in stone.

Indeed, I
would welcome
feedback
from readers.

My task was to transform the old code to the new machine. It turned out to be a major project. Every loop of code had to be carefully examined to understand its function. Every variable had to be defined. Each subroutine had to be analyzed for its purpose. How much easier things would have been had there been adequate documentation. Things would have been easier had there been footprints to follow.

To this end I would like to propose some standards. My standards are not chiseled in stone. Indeed, I would welcome feedback from readers. But in any case here is my idea for a standard to cover documentation.

1. Header

a. program name
b. author
c. date
d. compile information

2. Function

a. programs purpose
b. its usage
c. external routines called

3. History

a. version numbers
b. dates changed
c. modifications

4. Comment all variables

a. at declaration
b. or in separate section

5. Comment all the source code

a. comment major sections
b. cover unusual items

I think using the above format will help to document a program well. It will add to the code size. But it will leave valuable information about what was done. For the purpose of the magazine, I will probably shorten this format a little. In most cases, the programs appearing in the 68' Micro Journal are the first versions. Also, I try to cover the programs usage and purpose in the columns. For programs appearing here, I will uses a slightly shorter version. It is:

1. Header
2. Function -- brief
3. Commenting code and variables

Documentation is not just for the author. It is also for anyone that uses the program. I remember at meeting of a computer club that I was member, another member announced to the group that he found he could save time when transcribing programs from computer magazines. He simply left out the comments. Others started agreeing that this was a good idea. I sided with the few that felt that the comments were put in for a purpose. They were not put there to confuse and waste time. But rather to enlighten and save time.

I am not innocent!
Occasionally, I have paid for it.

A few years back I had the job of doing a rewrite of some code written in Basic.

My task was to transform the old code to the new machine.

How much easier things would have been had there been adequate documentation.

Documentation is not just for the author. It is also for anyone that uses the program.

Documenting doesn't stop with the program code.

Documenting doesn't stop with the program code. It is always helpful when archiving a program, to include a DOC file and a HISTORY file. Let's say we've just created a masterpiece of a program. It is called DOIT. Wherever you save programs, you create a directory appropriately called DOIT. Now in it you include the original source code, doit.c (or .BAS or .PASCAL or whatever you program type). You also include a working, debugged version. There should be a file called DOC that covers the function and usage of you masterpiece. You may want to go into greater details, then were covered in the source code. Finally, you create a HISTORY file. It covers the dates, version and modification history. Again, you may want to go into greater detail. You may want to add other files that prove helpful. A common one is README. It usually acts as an introduction to the rest of the directory. Put in whatever will be helpful.

On the system where I work, they encode the documentation in such a way that key parts can be later extracted. Useful information like the function and usage are removed and made into a separate file. That file is then added to a HELP directory. This can be very handy when you have a library of routines at your disposal. An entry to HELP with the appropriate name can elicit instant information.

You may not want to go the degree that I have outlined here. But I firmly believe that you can't document too much. (Take that with a grain of

salt, please!) With most programs being compiled, assembled or compacted, the comment lines are removed. So there is no argument that memory is being wasted. As for storage, disks are not very expensive. There is really no excuse for lack of good documentation. Just remember, write your program for someone else to understand.

REPLACING ANOTHER STANDARD COMMAND

I have been at it again. This time I am replacing the LIST command. I use it frequently. Most times I use it for listing things to the printer. Unfortunately, it does not do everything that I would like it to do. First, I find that when I dump something to the printer, invariably a line will get printed across the perforations where pages are joined together. Second, printing from top margin to bottom margin does not look very attractive when the sheets are later separated. And finally, if I list a number of files on one parameter line, they tend to run together. It is difficult to tell where one file stops and another begins on the printout. So, this month I offer a new version of LIST.

This version of LIST is written in C. I documented everything, according the framework, I outlined before. Hopefully this will make it easy to understand. I have included a functional and usage description. I also included the routines outside of main() and the first entry of its history. After each variable and constant is a comment of its purpose. Each main section of code includes a comment.

Another item, I left in is the debug code. Anything appearing between the lines #if DEBUG and #endif are used to debug the program. For example, in main(), I used printf to print the value of the flags and the files that are to be listed. While I was debugging the program I had a line in the beginning,

```
#define DEBUG
```

All the code was compiled with the program. When I was finished and everything was running the way I wanted it, I commented out the #define DEBUG. Then at compile the debug code is ignored. It does not appear in the final, executable module.

There are only a few external variables used. They are FFLAG, the form feed flag, PFLAG, the paginate flag, and HFLAG, the header flag. Initially, the flags are all cleared. They can be set from the input line by adding -f, -p or -h. If FFLAG is set, a form feed character is sent at the end of each listing. This is to prepare for another listing. Even if there is only one listing, it does advance the printer paper to the start of a new page. Setting PFLAG causes the top and bottom margins to be added. I set TOPSIZE to 6 and BOTTOMSIZE to 8. These seemed to be about right. Change them if you wish. The paper length is MAXLINES which is 66. This is the standard paper length. FFLAG, when set, causes a simple header to be printed. The header specifies the filename at the top of each page. This is helpful in identifying the files being listed.

Main() has only a few functions. It sets the proper flags depending on the parameters passed. If there are no file names that were passed, it prints a little help message. This is located in help(). Otherwise, the files are listed one at a time with list().

List() does the actual work. It opens the filename, if it can. Then it reads a line at a time and prints it to the standard output path. It keeps a record of the number of lines printed. If it comes to the start of a page and PFLAG is set, a top margin is printed. A bottom margin will be printed, when it comes to within 8 lines of the page bottom. If HFLAG is set, a header is printed at the top of each page. And setting FFLAG causes a form feed character to be sent after the listing.

There three other minor routines. There is ff(), which sends the form feed character. The routine cr() sends carriage returns for a specified count. And header() prints the file name at the top of the page. These functions are short and could probably be incorporated into the list() function. In fact, everything could probably be written into main(). The computer will process them either way. But for us humans, breaking up the program into parts makes it more understandable and easier to comprehend.

I also opted to use the low level file I/O. I use open(), writeln() and close(). I chose them since the file operations did not call for anything more

complex. I like the higher level file operations like fopen(), fscanf() and fclose(). They are more standardized and useful. They also add to the program's size. This version of list is about 2K in size. If the higher level disk I/O calls are used, its length grows to about 4K. My intention was to keep LIST short. And we need only the capability of the lower level calls.

One note about LIST. It has a macro defined for printing string data. It is:

#define print(c) writeln(OUT,c,strlen(c))

I use it to print strings. On the first pass the C compiler substitutes the string 'c' found in print(c) into the second expression, wherever 'c' occurs. OUT has been previously defined as 1, the standard output path. Actually, there is a small flaw with this construction. Think about this one and I'll tell what it is next time. I will give you two hints. The flaw won't harm LIST's operation in any way. And if you find and correct it, it will improve the program in some way. Think about it. I'll tell you next month.

That wraps up another month. I realize the idea of documentation is applicable to not only the OS-9 community, but to everyone involved in programming. I think it is an important aspect that cannot be ignored. My belief is (I hope it doesn't sound like I'm preaching) that we can show the rest of the world that we are professionals!

LISTING

```
 1 /* ****************
 2    Name:      list
 3    Date:      11-JAN-86
 4    Author:    Ron Voigts
 5    Compiling: ccl list.c
 6    ****************
 7    Function:
 8        This program replaces the standard
 9        LIST command.  It adds the feature
10        of doing a form feed between files,
11        adding margins at top and bottom of
12        page, and putting a header at the top
13        of every page identifying the file.
14    Usage:
15        list [-p] [-f] [-h] filename [...]
16            p=paginate, adds upper and lower
margins
17            f=form feed, advance paper after
printing
18            h=page header with file name
19    External routines (outside of main()):
20        1. help().....prints help message
21        2. cr().......output carriage returns
22        3. list().....list a file
23        4. header()...prints header for page
24    ****************
25    History:
26        1. Version 1.0   1-Jan-86
27    **************** */
28
29 #include <stdio.h>
30 #define FF "\x0C"              /* form feed
character */
31 #define OUT 1                  /* stadard
output path */
32 /* print macro */
33 #define print(c) writeln(OUT, c, strlen(c))
34 /* comments on next line during debugging */
35 /* #define DEBUG  */
36 #define MAXLENGTH 512         /* buffer size
*/
37 #define MAXLINES 66           /* lines per
page */
38 #define TOPSIZE 6             /* top margin
size */
39 #define BOTTOMSIZE 8          /* bottom margin
size */
40
41 char buffer[MAXLENGTH],       /* the buffer */
42      *s;                      /* character for
switch */
43 direct int fflag=0,           /* form feed
flag */
44            pflag=0,           /* paginate flag
*/
45            hflag=0;           /* header flag
*/
46 direct int count;             /* line count */
47
48 main(argc,argv)
49 int argc;
50 char *argv[];
51 {
52
53      /* set proper flags as passed in
parameter line */
54    while (--argc>0 && (*++argv)[0]=='-')
55       for (s=argv[0]+1; *s!='\0'; s++)
56          switch(*s) {
57             case 'f':
58                fflag=1;
59                break;
60             case 'F':
61                fflag=1;
62                break;
63             case 'p':
64                pflag=1;
65                break;
66             case 'P':
67                pflag=1;
68                break;
69             case 'h':
70                hflag=1;
71                break;
72             case 'H':
73                hflag=1;
74                break;
75             default:
```

```
                print("Illegal Option ");
                writeln(OUT, s, 1);
                print("\n");
                argc=0;
                break;
            }

/* check flag settings and file list */
#if DEBUG
{
   int i;
   printf("fflag=%d \n",fflag);
   printf("pflag=%d \n",pflag);
   printf("hflag=%d \n",hflag);
   for (i=0; i<argc; i++)
      printf("%s\n",argv[i]);
}
#endif

   /* print a help message if no files names
      were passed or process them */
   if (argc<1)
      help();
   else
      while (argc-->0)
         list(*argv++);
} /* end of main program */


/*
  A help message is printed
*/
help()
{
   print("Usage:\n");
    print("   list [-p] [-f] [-h] filename
[...]\n");
    print("      p=paginate, adds upper and
lower margins\n");
    print("      f=form feed, advance paper
after printing\n");
    print("       h=print header with file
name\n");
   print("\n");
}

/*
    The file is listed to the standard
output.
   On entry s points to the file name.
   The flags are processed according to how
   they are set.
*/
list(s)
char *s; /* file name */
{
   int path; /* path number */
/* print path name */
#if DEBUG
    printf("%s\n",s);
#endif
   /* open path to file */
   if ((path=open(s,_READ)) == -1){
      print("Can't Open ");
      print(s);
      print("\n");
   } else {
     while (readln(path, buffer, MAXLENGTH)
> 0) {
        /* do a top margin if needed */
        if ((count==0) && pflag)
           count+=cr(6);
        /* do a header if needed */
        if ((count==0 || count==TOPSIZE) &&
hflag)
           count+=header(s);
        /* print the buffer */
        print(buffer);
         /* the bottom margin if necessary
*/
        if ((count==MAXLINES-BOTTOMSIZE) &&
pflag)
           count+=cr(8);
        /* Reached the end of the page? */
        if (++count > MAXLINES)
           count=0;
      } /* end of while... */
       /* do a final form feed if necessary
*/
      if (fflag)
         ff();
   }  /* end of else */
   close(path);
} /* end of list() */

/* Subroutine prints EOL's for ct times.
   The count is returned.                */
cr(ct)
int ct;
{
   int t;
   t=ct;
   while (ct-->0)
      print("\n");
   return(t);
}

/* Subroutine prints a form feed to output
*/
ff()
{
   print(FF);
}

/*
   Print header for page, s points to file
name
*/
header(s)
char *s;
{
   print("File: ");
   print(s);
   return(cr(2));
}
```

EOF

SOFTWARE USER NOTES

A Tutorial Series

From Basic Assembler to HLL's

By: Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

Windrush Strikes Again!

What's new? If you are a regular subscriber to '68' Micro Journal, you have no doubt seen the ads from Windrush Micro Systems for their PLuS compiler, a 680XX code generating version of their PL/9 compiler. They have been selling a 6809 running version that runs under FLEX for several months. Now they have, or are about to have after testing by a few of their PL/9 users, a brand new version of PLuS for the 68000 running under OS-9/68K. I found an evaluation copy in my mail last Saturday. I couldn't do a great deal of fancy program checking because it was a limited capability demonstration copy. I have been promised a fully capable copy to test in about three weeks. By the time you read this, there will be an ad describing the product completely in the same issue. The compiler is along the line of that which has been advertised in '68' for some months, except that it is to run on a 68000 system as well as generating code for a 68000 or one of its variants.

The system includes a nice screen editor that is more than adequate for program editing, and a complete library of functions much like that provided with PL/9. There are a few additions to PL/9 provided as well. For example there is a 32 bit long integer type, and three specific unsigned types of data. My favorite shorthand statement of Pascal has been adopted. To assign a logical value to a "flag" previously one had to use:

```
if a < 0 then negative = TRUE else negative = FALSE;
```

Now the form used in Pascal is accepted:

```
negative = (a<0);
```

That is, the expression a<0 is evaluated and the TRUE or FALSE value of the result is assigned to the flag negative. PLuS requires the logical expression to be in parentheses. The other shorthand notation comes from "C". Rather than having to spell out:

```
index = index+1;
```

one can now use:

```
index++;
```

The equivalent decrement instruction index--; also works. The increment and decrement instructions also work on pointers to various size variables. The compiler keeps track of the data type and increments or decrements the pointer by 1, 2 or 4 bytes as required.

The manual is about half as large as the current PL/9 manual, but it is all there. Because of the large number of features that have not changed from PL/9 I was able to read through the manual in a couple of hours.

The test or demo programs that were supplied all compiled instantly and ran about as fast. I decided to try writing a program that would read an ASCII file of numbers and print them out. First I got the program running simply reading and printing the values as text strings, and then got bold and decided to try the conversion routines in the library files to convert the string to a REAL number binary format, and then reconvert a real variable value back to a string and print it out. I found a problem in the conversion of negative numbers. I have since spoken to Graham Trott, and he mentioned that this problem had been found and cured before I had reported it. We were able to make a temporary patch until the full version arrives.

These remarks are in no way meant to be critical of Windrush. Remember that I said this was an evaluation copy and as such really hot off the press, and I would expect to find a bug or two in either the software or the documentation. That is of course, the reason that smart software suppliers send out evaluation copies to people that they know will have applications for the software and will find bugs. A supplier can't envision all the uses to which their software might be put, and the best test is to send it to a number of users with diverse applications. I'll report more here as I get along a little farther with the testing and familiarization process on the fully capable version.

Autocad

Our company has had a version of Autocad, the computerized drafting software for some time. We first ran it in IBM computer version on a Tandy 1200 HD. It worked on that machine, but it was slow and a pain to wait for it to recalculate the graphics and update a screen. When the Tandy 3000 80286 system became available we tried Autocad on it and we believe there was a speed gain of about 5 times. The company decided to go all the way, and we have purchased a Sun 68020 based system with UNIX and a 70 Megbyte hard disk. The system is presently being installed and we expect a rather large increase in performance of the Sun system over the 80286. Autocad is not simple to use, but it is very capable and a beginner need not use every function right from the start. We expect to realize a great increase in productivity over doing mechanical drawings by hand,

though certainly not immediately. We have one problem with the system. After Unix and Autocad were installed and all the software that we are not going to use, was removed, we had 10 megabytes of user space left on the hard disk. We found that Unix reserves some 44 8K user buffer areas in RAM that we can't release for use even though the system will be single user. Six megabytes of hard disk space is reserved for task swapping that will never be needed. I think Unix is a little overkill for a small computer system!

The computer age is surely upon us. I submit in evidence the company for which I work, Hines Industries. We are a small company with about 50 employees. Offhand I can count thirteen computers that are in use every day, two or three of which have multiple terminals and users. The accounting department has a computer with multiple terminals, used of course for accounting payroll, etc. The Sales department has two or three computers, one of which is used with several terminals for the preparation of quotes, generation of sales letters, customer database files, etc. The others are used for word processing.

The mechanical engineering department / machine shop has one computer dedicated to the generation of programs for the CNC lathes and milling machines, and of course the new one for running Autocad. The service department has two for keeping customer service records and for generating programs for some of our standard products. The Software and electrical engineering departments have four in house and one loaned out presently. We have a Tandy 1200 presently being used to run software to program a large Modicon Programmable Logic Controller for a machine. We have the Tandy 3000 used virtually every day to run PC board layout software and Autocad for schematic diagrams, our old SWTPc 6809 system on which we develop software for our machines, a second SWTPc machine presently loaned out to a consultant who is doing some software development, and a home made 6809 based computer that is both a portable development system and a "test bed" for the computer hardware that goes into our product. The purchasing agent has a computer that he uses to generate purchase orders and keep track of inventory. That's a little more than one computer for every 4 employees!

PL/9 LIBRARY BUG

I strongly suspect that I have to take the blame for this one squarely on my shoulders. I seldom use the COS function of the SCIPACK.LIB that is part of PL/9, though I am responsible for writing it to a large extent. A week ago, at work, we had a program bug that seemed to be unexplainable. Eventually we traced it back to two things that we were doing that were unusual. First, since we normally display angle information in our system, we usually "normalize" all angles, that is, reduce them to an angle between 0 and 2PI radians or 360 degrees. Secondly, we don't normally include the whole COS procedure in our scientific function library, since the following will do as well, though not quite as accurately:

PROCEDURE COS(REAL OP); RETURN REAL (SIN(OP+PI/2));

The use of this trick reduces the code generated for the Scientific functions considerably, but that is beside the point. We tracked our difficulty down to an error in the COS procedure in the standard SCIPACK.LIB. It would seem that we adjusted the sign of the returned value nicely only if it happened to be in the second or third quadrant. If the angle were not normalized, as for example between 450 and 630 degrees, the returned value would incorrectly be Positive rather than Negative. The offending code follows with the fix clearly commented and added in upper case so it is visible.

```
   procedure cos(real op):
   byte negative, quadrant;
   if op = _pio2        then return real 0;
   quadrant = fix(int(op / _pio2));
   op = op - quadrant * _pio2;     negative = 0;

       QUADRANT = QUADRANT AND 3; /* ADD TO
NORMALIZE QUADRANT */

       if quadrant = 1 .or quadrant = 2 then
negative = 1;
   if quadrant and 1 then op = _pio2 - op; op =
_poly(op * op, .cos_coeff,5);
   if negative        then op = -op;
   endproc real op;
```

PRACTICAL MATHEMATICS

Something I was doing last week brought to mind a practical example of what one can do with a computer. I'll take an example that is a little more straightforward than the one I was working on, but the principle is the same. I don't know how many of you have studied Integral Calculus, but at least in my College days, it was not a subject that was studied by many people outside of the Engineering School. This bit might be a little insight into what Integral Calculus is all about, but even if that part leaves you cold, there might just be something practical here for you. Suppose for a moment that you want to find the volume of a cone shaped solid, fairly accurately. For simplicity let's assume the cone is 3 inches high, and 2 inches in diameter at the base. If we assume the base is circular, that implies that the base is perpendicular to the axis of the cone. The volume of such a cone is 1/3 PI * R^2*H. The diameter of the base is 2 inches so R, the radius is 1. PI*R^2 is therefore PI*1. Now H is 3, and we have to multiply by it and take 1/3 of the result, so that the result, that is the volume of the cone is just PI or 3.14159265 cubic inches.

Suppose I had forgotten the 'formula' for the volume of the cone, and also that I was weak in my Integral Calculus, but I remembered the basic principle of breaking the volume down into little pieces for which I could calculate a volume, and then summing all of them. Suppose we superimpose a coordinate system on the cone, putting the tip at the origin so that the cone axis lies on the positive X axis. Now let's break the cone up into a large number of very thin circular disks whose area we can calculate easily. Since the radius of the base is 1/3 the height, and the sides of the cone are straight, we can assume that the radius at any point on the X

axis is X/3. The area of a disk at that point on the X axis is PI*(X/3)^2. Now let's do a simple program in BASIC to solve for the volume. Assume we start at the tip of the cone and calculate the volume (area times thickness) for 3000 disks, each 0.001 inch thick. The Basic program uses a loop for X%=1 to 3000, and X=X%/1000.0, to calculate the radius at each point. It then calculates the area of that circle and multiplies by 0.001, the thickness, and adds it to the total volume V. You might note that the radius of the disk happens to be the radius at the largest edge of the disk. That is, the "edges" are not straight, but are part of the cone's surface. We might do better to take an 'average radius' for each disc, the value for the middle of the 0.001 inch thick disk. That improvement has been made and the sum of the volumes of those disks is found in V2 after running the program.

If you dump this program into your computer and run it under a good BASIC such as TSC Extended BASIC you will find that the value for V is a little high, being off by 2 in the 4th significant digit. The value for V2 is a much better approximation, having an error of 1 in the 8th digit. Of course, Extended BASIC has 16 digit arithmetic, and if you run this on a 6 digit BASIC, you won't get 8 digit precision. There are many calculations performed in running this program so be prepared to wait a few minutes for it to finish.

If we generalize this approach a little, call the height of the cone H and the radius of the base R, the calculation we have done is the basis for solving the integral to derive the general formula for the volume of a cone. Since I don't have an integral sign available for this, I'll spell it out in words. The integral we must solve is the Integral over the limits 0 to H of PI * (X*R/H)^2 dx. PI, H, and R are constants and can be taken out of the integral so we have PI*R^2/H^2 times the integral of X^2 dx. This integral is one learned the first week of an integral calculus course, and is X^3/3. Evaluating that over the range of 0 to H we obtain Volume = PI*R^2/H^2*H^3/3 -0.0 which simplifies to PI*R^2*H/3, which is the general formula for the volume of a cone.

The point of this is that it is possible to sum the volume of little slices of a larger solid to arrive at the total volume by using the computer. In the case of last week, the Integral that I would have had to solve was beyond my powers with Integral Calculus. I could set up the problem as the sum of the volumes of some thin slices, and doing so yielded a solution (though indirect and not precise) to my problem. If you use this technique and want to estimate the error in the result, reduce the thickness of the "slices" to half the original value, which should make the error about half as large, and note the change in the results. I did that in my problem of last week, and the results changed by about 1%, which indicated that the error in my second approximation was about 1%.

There are more accurate ways of doing digital integration. One good source for algorithms for such uses is the book "Some Common BASIC Programs" by Borchers and Poole, published several years ago by Osborne. The book contains programs for solving simultaneous equations, numerical integration and differentiation, fitting a curve, (approximating a function to fit empirical data using what is called polynomial regression), and a number of financial calculations as well.

Last Minute Notes

I just received an update on OmegaSoft Pascal OS-9/68K, and I will be reviewing it as soon as I get that latest version installed in the Mustang.

I also received a very interesting letter from Terry Ritter yesterday. Terry is one of the "creators" of the 6809, and he had some interesting observations about standardization, expending on my mention of the subject in my last column. I must get this off to '68' Micro Journal, but meanwhile I have sent a letter to Terry asking for permission to quote his letter here.

As of today, I still await the latest version of PLuS from Windrush. It is also an OS-9/68K version. I plan a review here as soon as I can do some testing of it as well.

```
10 REM VOLUME OF CONE 3" HIGH, WITH 1" RADIUS
20 REM PUT THE TIP AT THE X=0 POINT. AS X
INCREASES FROM
30 REM 0 TO 3, THE RADIUS OF THE CONE INCREASES
FROM 0 TO 1.
40 REM THEREFORE THE AREA OF A THIN SLICE OF THE
CONE AT A
50 REM GIVEN DISTANCE X FROM THE ORIGIN IS
(X/3)^2*PI
60 REM THAT COMES FROM THE AREA OF A CIRCLE
BEING PI*R^2
70 REM NOW SET UP A PROGRAM TO LOOP AND SUM THE
VOLUME IN A LARGE
80 REM NUMBER OF THIN SLICES OF THE CONE, SAY
0.001 INCH THICK.
90 FOR X%=0 TO 3000
100 X=X%/1000.0
110 V= V + PI*(X/3)^2*0.001
115 V2 = V2+PI*((X-0.0005)/3)^2*0.001
120 NEXT X%
130 PRINT"APPROXIMATE VOLUME IS: ";V
135 PRINT "BETTER APPROXIMATION: ";V2
140 PRINT"VOLUME BY FORMULA V=1/3 (AREA OF
BASE)*HEIGHT";PI
150 END
```

EOF

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01543

UNLIMITED MEMORY FOR THE 6809

Even though the 6809 can only directly address 64K of RAM at any one time, this does not really limit a program's memory use, if one makes use of the concept of virtual memory. Virtual memory looks like something for nothing, which is always suspect, but it actually works very well. What you are really doing is substituting disk storage for RAM storage. The most wonderful aspect of this is that it is essentially automatic in FORTH.

When FORTH was originally developed, RAM was very expensive and scarce, so there was an immediate need for a substitute. Even then, floppy disks were available at a cost effectiveness far beyond any other storage method. Therefore, the choice was consciously made to substitute disk memory for RAM memory. The only penalty paid for this exchange was in access speed, but an unlimited memory was obtained at this relatively small cost. Let's face it, FLEX disk I/O is SLOW, even when compared to CP/M. But it is still plenty fast enough to be comfortable. Furthermore, where else could you get multi-megabytes of random access memory for the price?

Virtual Memory in FORTH

Many people do not realize that they are using virtual memory from the outset with FORTH. Everytime a screen (block) is accessed in any way, that is an operation with virtual memory! Sure, it's a lot faster with a disk, but it works well with cassette tape. So don't be afraid of the concept of virtual memory, you have been using it all of the time--you just may not have noticed it.

The key word for making use of all of this available memory is BLOCK . The phrase <screen#> BLOCK puts an address on the Data Stack which points to a memory buffer; it also loads from the disk 1024 bytes of data from the four consecutive FLEX sectors which are indicated by the value of <screen#>. Of course, a FORTH such as from Sterns Electronics which uses a different number of bytes for a "screen" will use BLOCK to fetch that many bytes from the storage medium, but the idea is the same.

Virtual memory looks like something for nothing, which is always suspect, but it actually works very well.

Let's face it, FLEX disk I/O is SLOW, even when compared to CP/M. But it is still plenty fast enough to be comfortable. Furthermore, where else could you get multi-megabytes of random access memory for the price?

Reading Virtual Memory

There is nothing special involved in reading data from virtual memory, just use <screen#> BLOCK . You cannot use <screen#> LOAD or <screen#> LIST as conveniently, because of various side- effects, so be careful to use only the word BLOCK when you only want to move data from virtual memory into RAM.

LOAD does move data into the buffer, but FORTH also tries to interpret the data at the same time, and this could lead to all kinds of problems. LIST forces output to the display and/or the printer, which could also cause problems. So stick to BLOCK for all data access and you will stay out of trouble.

Writing to Virtual Memory

Writing to virtual memory is just as easy as reading it. Assume that you have some text that you want to save (remember that normal FORTH files are in ASCII) that presently is in RAM at a location called TEXT-BUFFER . Further, assume that you want to store this data in screen #57. To do this you must move the text to the disk with the following command:

'TEXT-BUFFER 57 BLOCK 'TEXT-SIZE CMOVE UPDATE FLUSH

'TEXT-SIZE is the number of bytes to be moved, and cannot exceed 1024 without overflowing the standard FORTH disk buffer. CMOVE does the movement of the data and the phrase UPDATE FLUSH causes the data to be written to the disk in the position corresponding to screen #57.

FORTH Files are not FLEX Files

I think that one of the biggest barriers for beginners is realizing that it really is this easy to read and write disk (or tape) data. It looks so simple that there must be a catch somewhere, but there is not. Well, maybe there is one catch, in that the operating system does not protect anything from being read or written over; that is the responsibility of the programmer. Truthfully, that is one of the things that I appreciate in FORTH--being given credit for knowing what I want to do. Of course, that also means that I have to accept responsibility for errors!

All of the FORTH systems that I know of which operate under FLEX give you the option of using the screen files in the FORTH style, or of using the normal FLEX text files. However, I don't know of any system which allows you to blithely mix the two forms in one file.

Wilson Federici's FF9 (FORTH-83), which I use, has several predefined words for handling FORTH files in the FLEX format, and I use the FLEX format quite often for finished programs. However, I found that this was not a very convenient way for me to develop programs, so I use the standard FORTH screens until the program is running the way I want it to, then I shift it to the FLEX format for regular use, since this is a much more secure filing system.

FORTH AND OS-9

Since I have no way to check what I want to say here, let me make sure that everybody knows that the following is just speculation.

I think I know why FORTH has not been mated to OS-9, Level I. The main reason is that there is really no need for it. If you are limited to 64K, then FORTH can easily stand by itself and, as I have shown above, does not really need a supporting DOS. Of course, it helps to have disks formatted to a commonly used DOS, but this bares no relationship to necessity. If somebody wanted to write a set of disk drivers, etc. to hang onto the FORTH backbone, there would be no need for FLEX or OS-9, except for the first time the disk was to be read. Look at those FORTH systems written to run with the CoCo "system" for a ready example of a fully functional system.

Several companies have advertised FORTH for OS-9, Level II, for the 68000. Again, mark well that I am only speculating, but I think that they may be using only 64K for the FORTH, and are treating it as if it were a stand-alone FORTH which just happened to be loaded from an OS-9 disk.

I have talked to several professional FORTH programmers about the possible limitations of 64K, and none of them have ever had to use more than 64K on any single machine. Of course, the whole system in FORTH to control the airport in Saudi Arabia uses more than 64K, but not in any one controller (as near as I can find out).

The basic problem, as I see it, in combining FORTH and OS-9 is that FORTH makes extensive use of absolute addressing, and OS-9 expects to use only position-independent code. Therefore, if you could fool OS-9 into thinking that the FORTH was in ROM, you might get away with using Level I. On the other hand, if you use Level II to assign 64K to FORTH by itself, it might look like pic to the OS-9, but absolute addressing to the FORTH. Can anybody shed any light on this? I have been asked by several people if I could point them to a FORTH for Level I, both for the CoCo and for controllers, but I have had to admit ignorance.

If I were better versed in OS-9, I would try to adapt to FORTH the OS-9 disk drivers (SDISK and BOOTFUX) sold by D. P. Johnson. This way, FORTH could read and write disk files on OS-9 formatted disks. If we could do this, then the fig-FORTH could be readily adapted to run with (not on) an OS-9, Level I system, and I think that FF9 could also be converted. Don't write to me asking for more details, since I cannot do it; I am only suggesting that someone more skilled in OS-9 could try it. If it works, there might be a small market; you won't get rich, but you might make expenses while contributing to the general welfare.

STRING CONSTANTS

I have discovered a neat programming trick which I would like to pass along. It should work with any kind of FORTH for any CPU; it works with LMI Z-80 FORTH as well as with FF9.

Some versions of FORTH for other CPUs have the capability of compiling string constants by enclosing the string with quotation marks or by using the ," <string>" combination (FF9):

```
CREATE TEST-STRING ," This is a test"
```

If you type TEST-STRING COUNT TYPE , the string will appear on the display. Obviously, this is of no value to anyone, but if you have need for default strings for use in a file, this is a good way to generate them. Since I had such a need, and a need to be able to print out the string to show the data-entry operator what the default answer would be, I badly needed a reliable way to construct a long series of string constants.

This is when I realized that a definition using the ," pair could solve both my problems. I defined a printable string in the conventional way:

```
: .TEST-STRING ," This is a test" ;
```

I could then use the definition to show the clerk the default answer by simply including .TEST-STRING in the appropriate word.

Since the FORTH word ' (tick) puts the compilation address of the next following word onto the Data Stack, I only needed to add the necessary offset to this address in order to have a pointer to the first character in the desired string. Furthermore, this offset must be constant for all such definitions, so I had my reliable pointer to the string. In order to put this string on the disk I needed to use the following phrase:

```
' .TEST-STRING 5 + 27 BLOCK 14 CMOVE
UPDATE FLUSH
```

This resulted in having the desired string stored onto the disk in screen #27.

The important key to this technique is finding out the amount of the offset, which was 5 in my case. I expect that it would be 5 in practically all renditions of FORTH, but you should use DUMP on a suitable definition to verify this. Whatever you find this offset to be, it will always be the same until you change to a different FORTH.

I also found, by using DUMP . that there was only a 2-byte penalty for using ." instead of ," . Therefore, I really got "something for nothing", since I was able to make one definition do the work of two! This certainly qualifies as a programming trick, since it cannot be guaranteed to be transportable, but it must be reliable if it can be made to work for you at all.

CONSTANT versus LITERAL

I want to pass along another tip I picked up at the local FIG meeting. There was a group discussion about what makes up good FORTH programming practices, and several suggestions were offered which I hope to describe in detail. However, one suggestion was so good and so simple that I want to pass it along now.

One member (a FORTH professional) suggested that whenever you were looking for speed in a definition, you should never use a LITERAL . He had found that he could make a program run faster if he defined all of his LITERAL numbers as CONSTANTs. I tested this when I got home and found that he was right! With a "bench mark" type of program, a definition had a measurably faster execution time if a number was predefined as a CONSTANT outside of the definition, rather than being used simply as that LITERAL number inside the definition. For example, if you need to use the number 5 within a definition, first define it this way:

```
5 CONSTANT 5
```

This is similar to the old BASIC axiom that all numbers used more than once should be assigned to a variable at the first part of the program. We all know this, but it is easy to forget.

The programmer who suggested this at the meeting further said that he now automatically defines all of the numbers 0-255 as CONSTANTs in all of his long programs, since he probably uses all of them at least once in any large programming effort. Probably, we who are amateurs don't need to go to this extreme, but our programs will run faster with judicious use of this idea.

MORE ON STATE-SMART WORDS

I received a letter from Wilson Federici the other day in which he rightly took me to task for giving the impression that FORTH- 83 does not permit state-smart words. I did not mean to sound like that in my complaint in the December, 1986, column; but I did come across a bit more strongly than I intended. State- smartness is definitely acceptable in FORTH-83; I just got carried away in trying to make a point about some words being changed in a way which made them less convenient to use.

BOOK REVIEW

Mastering FORTH Anita Anderson & Martin Tracy Brady Communications Co., Inc., New York, 1984 ISBN 0-89303-660-9

In retrospect, I guess that I was too hard on Sterns Electronics for having such meager help for the FORTH novice in the instruction manual accompanying Color-FORTH. I say this, not because I have decided that the manual is adequate, but because I have since had an opportunity to examine some other commercial offerings, and I found those manuals were no better.

Therefore, it is necessary for the FORTH novice to plan on buying at least one textbook on the basics of FORTH programming. I think that I have found the one that I would recommend as being the best for beginners: "Mastering FORTH". It is oriented toward the FORTH-83 dialect, but is really useful for any version of FORTH.

"Mastering FORTH" covers in adequate detail all of the required words found in FORTH-83, but goes on to cover most of the others which the standard lists as optional. Each word is described in the context of its use, so that the examples do a very good job of clarifying some of the more obscure definitions. The authors do a particularly good job of explaining the meaning of state- smartness.

They also cover the obvious requirements of explaining RPN math and the concept of Data (Parameter) and Return Stacks. The explanation of how a definition header is organized is very useful, even if your FORTH has its headers organized in a different order; at least you can learn why FORTH headers work.

Each chapter ends with programming exercises, with answers, which show you whether or not you really understand the contents of the chapter. Most of the exercises produce useful utility definitions or program fragments which can be used in practical programs.

The only problem with the book is that its discussion of the editor is too limited. It describes the use of a screen editor, which is not generic. It appears to be tied too closely to the MicroMotion editor, so may not be of much use to 68xx readers. But, the description of the general operation of a FORTH editor will be of help as a supplement to the editor instructions with your own FORTH version.

Also, you must be careful when reading the section on disk files, since the authors assumed

that the whole world is bound by CP/M, MS DOS, or PC DOS. The section on files is still useful, you just may not have all of the possible words they refer to.

All in all, I rate "Mastering FORTH" very highly, and recommend it, particularly if you plan to buy only one book on FORTH.

Fig 1.

```
#BUFS     ( -- n )
        A CONSTANT which returns the number
disk buffers. The default is 2.

>BUFS     ( -- adr )
       A CONSTANT which returns the address
of the first disk buffer( fig-FORTH  FIRST ).

BLK  ( a USER variable )
        Contains the address of the BLOCK
being interpreted.

BLOCK     ( u -- adr )
         Returns the address of the disk
buffer to the Data Stack and reads the data
from block "u" to "adr" if it is not already
there.

BUFFER    ( u -- adr )
       Same as BLOCK , but the disk buffer
contains garbage.

BUFSIZ    ( -- n )
        Returns the total number of bytes
required for a disk buffer.

DR0  ( -- )
DR1  ( -- )
      Selects the appropriate disk drive by
preloading OFFSET .
       The value in OFFSET is added to BLOCK
number to allow for this selection.

DSKERR    ( a USER variable )
        Error code from last execution of
RDWT . Defaults to 0.

EMPTY-BUFFERS  ( -- )
       Mark all blocks as empty. FF9 fills
all buffers with BL . Does not write UPDATEd
buffers to the disk.

FLUSH     ( -- )
      Executes SAVE-BUFFERS and deallocates
all disk buffers.

LIST ( n -- )
      Display the ASCII text of screen "n".
Load "n" into SCR .

LOAD ( n -- )
       Interpret screen "n". Interpretation
will stop at the end of the screen.

OFFSET    ( a USER variable )
        Contains the offset added to the
block number to cause the selection of the
proper disk drive.

PREV ( a USER variable )
        Contains the address of the most
recently used disk buffer.

R/W  ( adr u ? -- )
      Executes RDWT and stores error code
in DSKERR . Aborts with message on error.

RDWT ( adr u ? -- error-code )Access codes
for R/W :
            ? = 0:  Writes data at "adr" to
disk block "u".
             ? <> 0: Reads data from disk
block "u" to "adr". Returns error-code or 0
for no error.

SAVE-BUFFERS   ( -- )
        Write all UPDATEd buffers to disk.

SCR  ( a USER variable )
         Contains the screen number most
recently referenced by LIST .

UPDATE    ( -- )
      Mark last-referenced block as having
been modified.
```

Figure 1: FF9 disk words.

Fig 2.

+BUF	Not used in FF9.
?LOADING	Not used in FF9.
B/BUF	Similar to BUFSIZ in FF9.
B/SCR	Not used in FF9.
BLOCK-READ	Not used in FF9.
BLOCK-WRITE	Not used in FF9.
FIRST	Equivalent to >BUFS in FF9.
MESSAGE	Not used in FF9.
USE	Not used in FF9.
WARNING	Has a different meaning in FF9.

Figure 2: fig-FORTH disk words at variance with FF9.

EOF

SOFTOOLS

By: Art Weller

A software bonanza!

Everything you might need

more than two dozen excellent utility programs based on the book "Software Tools in Pascal)" by Kernighan and Plauger.

adapted to the **FLEX9 environment added further enhancements a very usable package for **FLEX users

you just can't do without!

A software bonanza! That's the only way to describe the SOFTOOLS program set by Martin Gregorie. Everything you ever thought you might need to work with text files, consisting of more than two dozen excellent utility programs based on the book "Software Tools in Pascal" by Kernighan and Plauger. Martin has adapted these to the **FLEX9 environment and added some further enhancements that make these a very usable package for **FLEX users -- you are certain to find something in this set that you just can't do without. Many of these are nice enough to use as the basis for a separate software review, but I doubt that Don will want to double the size of this issue, so I'll have to just hit the highlights in a summary.

To start at the beginning, the set includes a very good "READ-ME" file that contains all the needed instructions to get set up and running. The most important of these deal with the matter of printing out a hard copy manual for the SOFTOOLS set. Since the set includes a text formatter (more on it later), it creates its own manual. As supplied, the documentation prints out to a printer having a "compressed" mode of 132 characters per line. But if your printer does not, there are instructions on how to muddle through anyhow. An initial chore in getting set up is to process a configuration program that generates a file of parameters for your printer and terminal display unit. This file is then appended to the programs that need it (HEX, PAGE, SCREEN, TPROC) thereby "customizing" them to the requirements of your system. This need only be done once, unless you later change hardware. This is just as well, as I found it to be the exception in that CONFIG was rather tedious to use.

Having printed out the user manual, you will see that each program has been given its own section and the how-to-use is clearly and explicitly presented, including examples where this is needed to help clarify meaning.

Before getting to individual programs, a few comments that apply to all are in order. All have been compiled using *PL/9, so are more compact and faster than otherwise expected. All that use command line parameters have a default mode in which omission of the parameters will provide a small "help" display showing the syntax or brief explanation as a memory jogger for programs that might be used infrequently. *PL/9 source code is available and in itself constitutes a valuable library of *PL/9 procedures. A degree of redirection is provided so that input might be from a file or from a keyboard and output to VDU, file or printer. The latter is further enhanced by switching parameters automatically when going from screen to printer (ie a "page" changes from 23 lines to 66 lines, assuming that's how you originally configured), so that all output gets a degree of formatting. **FLEX pause and TTYSET parameters are properly handled, too.

Several of the programs incorporate a pattern recognition feature that is more versatile than most. The usual "wild card" is allowed and also a range of characters may be specified. For example, you can search for lower case alpha [a-z] or a sub-set of lower case [m-p], etc. An "escape character" is included to permit, for example, command line characters that **FLEX would otherwise discard (eg space or comma). Further, matching may be constrained to the beginning of

a line or end of a line (or NOT at the beginning or end). A feature that is difficult to express in a few words is that this wide range of options can be used to define "fragments" up to nine of which may then be used to construct a "pattern" for a search or replace operation giving the user an impressive degree of **FLEXibility that is at first overwhelming. Rest assured that pattern matching is covered more extensively and clearly in the SOFTOOLS manual than is this quick overview. Moving on to discuss these programs more specifically, the first makes use of the "pattern" feature.

CHANGE <infile> <outfile> +from to

This is a function familiar in editors of all kinds, but CHANGE works at the command line level and on a whole file (globally). Assume you are trying to use some old routines in a new program and find some of the variable names clash. Just CHANGE them!

CLEANTXT <infile> <outfile> +v

This converts text files to standard **FLEX format by stripping non-printing characters. It's a "smart" conversion in that the <CR> is retained, <BS> deletes a prior character and <CAN> kills the current line (where these are defined in TTYSET). A (v)erify option displays each line changed and reports on the changes made.

COMMON <file1> <file2> <outfile> +n keyfields

Compares the contents of two input files, reporting the results in the output file as to whether the lines matched or not. Obviously the pair of input files must be in order. The match can be modified by parameters in up to 10 key fields. This program could be used to build your personal dictionary and to compare text files against it for spelling.

COMPARE <file1> <file2>

A line-by-line comparison between two files in which identical lines are passed over and mismatched ones listed to screen.

CONCAT <outfile> +<input file list>

Similar to the **FLEX APPEND, this forms several files into one sequentially, but can also be used to list files to screen. Note the difference in command line sequence.

DOCUMENT <list-of-input-files> <outfile> +fp

This is a text file oriented program specifically designed for use with *PL/9 source files. (You all use *PL/9, don't you?). Input files are, obviously, the *PL/9 source and the output is the extracted lines. Default is to list the first line of the program file to terminal, but option "f" will suppress this, while option "p" extends the output to include procedure processing. I found it to be very useful for examining the parameter passing aspects of procedures.

ECHO <outfile> +<argument list>

Similar to a **FLEX program of the same name, this will echo the argument list to screen, but if the optional outfile is included, the echo is to file.

FIND <infile> <outfile> +pattern n

If you have never used a "find" command, the description of this is not likely to impress you. If you have, then you'll appreciate the improvement that is provided by use of the "pattern" matching I mentioned in the introduction above. Patterns can be constructed to make the matching as selective as needed, optimizing between being flooded with matches, and missing the desired one. You can use wild-cards, be case sensitive (or insensitive) or use FIND to extract lines to a file for use in another application. One of the best file searching tools I've used.

HEX <infile> +n

Displaying the bytes of "infile" in hexadecimal and ASCII, this is a file dump program like many others; what more can I say. But everybody needs one. The option "n" permits skipping the first "n" bytes in a file, as a time saver.

INCLUDE <infile> <outfile> +l

This is another file manipulation program that is primarily oriented toward *PL/9 files, though it could be used with others. The default is to merely copy infile to outfile, but if a line contains "include filename" then the program switches to copying "filename" after which it returns to "infile". Any number of "includes" can be used within the source file. If the "+l" option is specified then "filename" is not actually copied, but the outfile will show each "include filename" indented so as to show hierarchy structure. The primary application would be to generate an output file that contains the complete listing of all the code needed by a *PL/9 program.

KWIC <infile> <outfile>

This is an acronym for Key Word In Context, and operates on a text file by "folding" lines so that each word is rotated one at a time, to the beginning of the line until all the words have been so shifted. By itself, this is of little value, but used in conjunction with other tools in this set (such as SORT) allows for a detailed examination of the file's structure.

LISTDIR <drive> <outfile> +hd

A substitute for CAT, this is a minimal directory listing. In all honesty, if you have Leo Taylor's CAT command, you'll not be impressed with this one. Included for completeness.

MEMSORT <infile> <outfile> +keyfields

A high speed text file sorter with lots of options. Case of alpha characters may, optionally, be ignored and sort may be in ascending or descending order. As many as 10 sort fields may be specified. File sizes are limited by memory constraints, but see SORT for larger files. MEMSORT is very fast.

MULTICOL <infile> <outfile> +h=0 w=79 c=2 d=20

If you work with text files, you must have this. How many times have you had information that you thought would look better listed more than one column to a page, but which you wanted to read down rather than across. A

very difficult chore with most editors, but extremely simple with MULTICOL. Space for page headings may be reserved, the width of columns and the number of columns may be specified. Lines per page may range from 5 to 78.

PAGE <input> +nw=80 d=66

This prints text files with page headings and is very similar to the familiar LIST, but with added options of specifying the page width and depth on the command line. Before saying "ho-hum" you should remember that this is one of the "smart" programs noted above that "knows" whether it is talking to the screen or to the printer and adjusts output accordingly, to include issuing appropriate printer control codes.

REMOVE <file list>

A fast disk file deletion program, this removes the files specified on the command line. A "watch out"; no prompts are issued, it just zaps them!

SCREEN <infile> +n

Another list utility, this conforms with the parameters specified during the configuration session. Word-wraps text to match your screen width and pauses for each screen full. Screen depth may be altered at run time using a command line option.

SORT <infile> <outfile> +keyfields

A real gem, this is a super version of MEMSORT mentioned above. Still has the ascending/descending sort option, up to 10 keys, case over-ride, sort on nth word and sort on n characters and, if the file is small enough, sorting is done in RAM. But files up to your full disk capacity can be handled by SORT. The large files are automatically sorted into a series of smaller files stored temporarily on disk, then these are merged into a final sorted file.

TPROC <infile> <outfile> +s=0 e=10000

A small, but competent text formatter, TPROC is used very effectively in producing the printed documentation of the SOFTOOLS user manual. Formatting directives are identified by a period (full stop in the UK!) in the first character position of a line, then two or more characters identify the desired action. Text may not be included on these lines. Headers, footers, indenting, paragraphing, justification, page length -- in fact a full complement of formatting options are available as are a subset of printer command options. No "macro" capability is included, nor is "mail merge" supported, but input from the terminal is allowed and the ability to "include" specified external files into the runstream give enough **FLEXibility to perform these functions in a low volume environment. Notice that this, too, is one of the "smart" programs, outputting printer control codes only when actually sending to the printer and even checking the line width, then resetting the printer to compressed mode if needed.

TRANSLIT <infile> <oputfile> +fromset toset n

While reading the input, character by character, TRANSLIT converts text that matches "fromset" to "toset" in the output. Line numbers may be, optionally, added to the output. More than just a straight character substitution, applications may be hard to visualize. So to give an example, suppose all the words in a text file were to be broken out on separate lines by adding a <cr> after each. Then by sorting and removing duplicates this could easily form the basis for a personalized dictionary for use with a spelling checker.

UNIQUE <infile> <outfile> +x keyfields

After the "infile" has been sorted, UNIQUE may be used to check for duplication between adjacent lines. A single "unique" line will then be output. Optionally, the number of duplicates may be reported (as in a word count) and the comparison may be based on up to 10 keyfields with key field specifications including all the same **FLEXibility as with SORT. It could also, obviously, be applied to an unsorted file to learn if any adjacent lines were identical.

UNROTATE <infile> <outfile>

Used in conjunction with KWIC, UNROTATE reads the input file a line at a time and outputs an "unfolded" version in which each line has been formatted to display each word, showing the context in which it is used. If the input file has been sorted, the words will be presented in sequence.

WC <infile> <outfile> +w

An acronym for "word count", WC will count the "words" contained in the input file; where "words" may be characters, lines, or words. That is, a sequence of characters separated by a space, tab, or <cr>. A word count is the default, with "lines" or "characters" optionally with a +L or +C. The infile is not listed to the output, only a report of the number of items found. Notice that the definition of "word" includes things we don't usually call words; for example, a date or a number.

There they are. I can't resist the temptation of commenting that the conversion of these programs to *PL/9 makes them noticeably superior to the originals in Pascal. They are more compact, using less code (and, of course, no run-time interpreter) and a lot faster.

By:
Art Weller - El Paso, Tx

* It should be noted that PL/9 is not necessary in order to run the programs. PL/9 was the compiler used to compile the programs to object code. They run as any other ".CMD" command file.

**In most instances references to FLEX will also apply to SK*DOS.

EOF

XBASIC Xplained

or

Things you won't find in the documentation

Introduction

XBASIC is a language which enables the "speaker" to communicate with his computer. He can restrict its use to simply persuading this alien device to print "Hello", or become so proficient in the more subtle points of its grammar that he can almost train it to sit up and beg on command. The article supplied by TSC to purchasers of XBASIC, although very well written, is really equivalent to a high-level dictionary wherein one may find definitions of each and every word in the language. It's not meant to be a tutorial on how to program in the BASIC language, though certain aspects are covered in some detail, in particular the section dealing with data file structures and how to move data in and out of those files. Just as with any other natural language, however, a dictionary may enable one - with some effort - to construct sentences of the order of "Please, to bathroom this is way, no?", but will hardly suffice to carry on a sophisticated scientific discussion. So too with TSC's article, which admirably suits its purpose of giving a newcomer a feel for the language and getting him going. In this series we hope to pick up where they left off.

Several years ago, just for the fun of it, I spent quite some time disassembling BY HAND an early version of FLEX2 XBASIC, and then, about the middle of 1985, I had need to use this as a reference for FLEX9 XBASIC in order to incorporate into the language a new command, "EDIT", whereby I could edit any line in a program - without having to re-type the whole line in case of error. This led to my submitting to 68 Micro Journal a series of letters explaining in some detail my experiences with various aspects of the language. In tum, this produced correspondence with readers all over the world, prompting even further submissions over quite a long period.

The time is now ripe, I think, to gather all these letters into one compact tutorial for the benefit of newcomers to the 68xx(x) microcomputer scene, and hopefully some of the expanded explanations may be useful even to those readers who have earlier read my original series of letters.

SOME COMMENTS

Before I get down to the "nitty-gritties" though, I would like to give some advice to those who have tried writing a not-so-small program in XBASIC. You see, there are several schools of thought on the best way to write such programs - - some people, mostly theoretical types, will advocate a system known as "top-down", others "bottom-up", others won't even consider a program that isn't "structured", while others would rather die than use that much-maligned word 'GOTO' in a program.

We have no time for such concepts - what we wish to do here is to make XBASIC easy and painless to use. Then, when we have a thorough grasp of its principles, we can take a look at those other ideas, and make a reasoned judgment, based on knowledge, as to whether they will serve any useful purpose FOR US. No matter that the other guy swears that his is the only way to true salvation! To each his own!

Continued on page 38

19. SWAP -- swaps any two variables
20. SYSCALL -- system call
21. UPPER -- converts a string to uppercase

For OS-9 - $44.95 - Includes Source Code

See Review in January 1987 issue 68 Micro Journal

SOFTOOLS

!!! NEW PRODUCT !!!

The following programs are included in object form for immediate application. PL/9 source code included for customization.

READ-ME Complete instructions for initial set-up and operation. Can even be printed out with the included text processor.

CONFIG one time system configuration.

CHANGE changes words, characters, etc. globally to any text type file.

CLEANTXT converts text files to standard FLEX, SK*DOS files.

COMMON compare two text files and reports differences.

COMPARE another check file that reports mis-matched lines.

CONCAT similar to FLEX, SK*DOS append but can also list files to screen.

DOCUMENT for PL/9 source files. Very useful in examining parameter passing aspects of procedures.

ECHO echos to either screen or file.

FIND an improve find command with "pattern" matching and wildcards. Very useful.

HEX dumps files in both hex and ASCII.

INCLUDE a file copy program that will accept "includes" of other disk files.

KWIC allows rotating each word, on each line to the beginning. Very useful in a sort program, etc.

LISTDIR a directory listing program. Not super, but better than CAT.

MEMSORT a high-speed text file sorter. Up to 10 fields may be sorted. Very fast. Very useful.

MULTICOL width of page, number of columns may be specified. A MUST!

PAGE similar to LIST but allows for a page header, page width and depth. Adjust for CRT screen or printer as set up by CONFIG. A very smart print driver. Allows printer control commands.

REMOVE a fast file deleter. Careful, no prompts issued. Zap, and its gone!

SCREEN a screen listing utility. Word wraps text to fit screen. Screen depth may be altered at run time.

SORT a super version of MEMSORT. Ascending/descending order, up to 10 keys, case over-ride, sort on nth word and sort on characters if file is small enough, sorts in RAM. If large file, sort is constrained to size of your largest disk capacity.

TPROC a small but nice text formatter. This is a complete formatter and has functions not found in other formatters.

TRANSLIT sorts a file by x keyfields. Checks for duplications. Up to 10 key files may be used.

UNROTATE used with KWIC this program reads an input file and unfolds it a line at a time. If the file has been sorted each word will be presented in sequence.

WC a word count utility. Can count words, characters or lines.

SPECIAL INTRO PRICE:

FLEX - SK*DOS

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.

F, S and CCF, U - $25.00, w/ Source - $50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only - OS-9 Regular $149.95

SPECIAL INTRODUCTION OFFER *$69.95*

DISK UTILITIES

OS-9 VDisk from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.

Level I OS-9 obj. $79.95; w/ Source $149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX, SK*DOS Format so it can be used normally by FLEX, SK*DOS; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX, SK*DOS Directory, Delete FLEX, SK*DOS Files, Copy both directions, etc. FLEX, SK*DOS users use the special disk just like any other FLEX, SK*DOS disk

O - 6809/68000 $79.95

For instance, my friend Denis and I recently wrote a MONOPOLY program for our GIMIX 6809 machines. How did we go about it? Well, our first step was simply to get the playing-board to display on the screen (using the graphics capability of the GIMIX 80x24 Video-Board). Then we persuaded it to display two playing-pieces, represented by a Knight and a Pawn, on the GO square. Notice, no in-depth study of the entire game program, no "top-down" or "bottom-up", and what's even worse, we had GOTOs all over the place. But ... the program worked, and was fairly easy to understand. The way we figure it is -- If you can't even get it to display the Game-Board, then you're wasting your time planning the rest of the program! Next, we got our picture of the dice to change spot-patterns on receipt of an 'R' command (for ROLL), and the appropriate piece to move correctly to its destination. No buying of properties, or collecting of rents yet. Those were the next steps. And so we proceeded, one stage at a time, until our program grew to the point where we could actually play a 2-player game, with the computer acting as Banker and verifier of valid moves. Some people may claim that this is "top-down" or whatever, but we prefer to think of it as "one step at a time" with no particular sequence in mind. Sometimes we might design the end-game before the centre portion!

I should, perhaps, take time out here to explain that Denis and I are not really interested in playing the games once we've developed them, though some of our visitors do enjoy our Backgammon, Cribbage or other relaxations. No, the fun is in the challenge of seeing if we can actually get our computers to do what we want them to do. Once done, we move on the next game, or utility, or whatever. Sure beats doing crosswords as an intellectual stimulation, and certainly a lot less painful than struggling to the top of Mount Everest only to climb down again once you get there.

THE INNARDS OF XBASIC

OK, enough of that for now. Let's begin with a brief look at the 'innards' of the workings of XBASIC, keeping in mind that any HEX addresses given will obviously vary from one version to another, the latest, and possibly final, version for the 6809 being Version 24.

We'll assume that we've just keyed in the following short program in response to the READY prompt :

```
 5 A=3
10 B=5
 7 C=9        in the exact order shown.
```

A simple LIST command would display the lines in correct numerical order, while an examination of the same program saved to disk (using a program such as DISKEDIT) would show precisely the same thing. So it would be quite natural to suppose that this is also the way that XBASIC actually stores the program in memory during a LOAD or keying-in operation, though some people already know that in some way XBASIC structures its lines so that each line carries a pointer to the next one in sequence, with the final line's pointer being 0000, to indicate end-of-program. Here's what we'd actually see if we performed a dump of the program-memory for our little program, the addresses shown being those of the original FLEX2 XBASIC :

```
Address  4B05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13
 Data      00 05 4B 23 7E 80 00 00 69 A4 00 03 01 33 7D

 Address  4B14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22
 Data      00 0A 00 00 7E 80 00 0B 69 A4 00 05 01 35 7D

 Address  4B23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31
 Data      00 07 4B 14 7E 80 00 16 69 A4 00 09 01 39 7D

 Address  4B32 33 34 35 36 37 38 39 3A 3B 3C
 Data      80 41 00 00 00 00 00 00 00 00 00
```

```
Address  4B3D 3E 3F 40 41 42 43 44 45 46 47
Data       80 42 00 00 00 00 00 00 00 00 00

Address  4B48 49 4A 4B 4C 4D 4E 4F 50 51 52
Data       80 43 00 00 00 00 00 00 00 00 00
```

Who would have thought that such a simple 3-line program could lead to so much code? And what does it all mean? Let's take it a little at a time, just as we do for program development.

First of all, it's quite obvious that the first 3 lines of code represent our actual program, as we very readily spot our line-numbers 5, 10 and 7 (converted to HEX) in the first two bytes of each line. In no time at all we see that the next two bytes contain the ADDRESS of the next line IN NUMERICAL ORDER, not the order in which we keyed them in and not the actual line-number itself. Thus they serve as pointers to the precise address at which the next line in sequence is stored, except for the last line, whose pointer is 0000.

So far so good, but what about the '7E' in column 5 of each of these three lines? This is XBASIC's way of knowing that the actual string of data in the program-line is separated from its line-number by a single space. If the statement part of the line were indented by more than one space, a '7F' would be stored here with an extra byte after it to indicate the number of spaces. So, you see, as far as program memory usage is concerned, only one extra byte is needed - no matter how deeply you indent your lines beyond one space. The same codes are used to keep track of other SPACEs encountered in the program-line, such as 'X = 5' rather than 'X=5'. Not so when it's saved to disk, however.

Column 6 of these lines contains an '80', which tells XBASIC that the line begins with the name of a Floating-Point variable (henceforward we'll abbreviate Floating-Point to FP). An '84' would signify an integer %-variable (in the range +/- 32767), and an 81 a string $-variable. The next 2 bytes indicate the displacement of the Variable itself in the Variable-Stack - composed of the 3 shorter lines below the program-lines. For instance, the first Variable 'A' has a displacement of 0000 (ie, it's the first Variable in the Stack), where we see its symbol 80 repeated, followed by its 'name' in ASCII (that is, A=41). Actually, 2 bytes are reserved here, in case we used a 2-lettered Variable-Name, such as AX or J5, with an additional 8 bytes reserved for the current numerical value of the variable - initialised to zero. As the program executes during RUN, XBASIC keeps the value continually updated. Similarly, the Variable 'B' commences 000B bytes in (that's 11 in human terms), and 'C' is 0016 bytes in (22 to us). Notice that a program-line does not itself identify the variable by name; it simply says "Here is a FP, or Integer, or whatever, displaced by so many bytes in the Stack - if you want to know more, then go to the Stack for full details!"

Now back to our program-lines. '69' is XBASIC's "token" for the '=' sign which follows the Variable-Name. More on Tokens later, and how you can speed up XBASIC a little by modifying its Command-Table and Statement-Table. For now, let's move on to the next byte, an 'A4'. Does this mean that all our variables are equal to A4? Not at all! This is simply a code to indicate that each is equal to a CONSTANT in the range +/-32767 (whose value follows) and not to another Variable - as in the statement A=X, for example. The value (in HEX) appears in columns 11 and 12, and again in ASCII in columns 13 and 14, the 01 indicating 'one digit', followed by 33 for Line 5, 35 for line 10 and 39 for Line 7. These, of course, are the ASCII values for 3, 5 and 9 respectively. Finally, each line ends in a '7D', which is XBASIC's token for a CARRIAGE-RETURN.

The assignment-codes for Variables depend on the bit-values in a single-byte code. For those interested in the details, here is a table showing the meanings attached to the individual bits, as far as I've managed to decipher them to date :

Bit 7 A variable of some kind, as distinct from a Command or Statement (whose Tokens all lie in the range 00 - 7F).
Bit 6 A pre-defined variable (such as FN)
Bit 5 A Constant
Bit 4 Subscripted
Bit 3 Not deciphered yet
Bit 2 % integer in the range +/-32767
Bit 1 Line-Number
Bit 0 $ string

Thus the code for a Line-Number such as the '1000' in 'GOTO 1000' would be A2. Converting A2 to binary gives 1010 0010, and thence (reading the '1' bits from left to right) we get (i) Variable (ii) modified to Constant and (iii) Line-Number. It may seem strange to have a Variable-Constant, but the word 'Variable' is used solely to distinguish the code from a Command or Statement code (token).

I would be more than pleased to hear from anyone who has managed to take the above deciphering a stage further, or who can confirm Bits 6 and 4.

At this point, I would suggest that you experiment a little yourself by first of all setting your free memory to an all-zero state, then loading XBASIC and writing a short one- or two-line program. Good examples are A=X, or A%(1)=19, or better still A%(5,11)=15. The Memory-Examine function of your 'BUG' (be it GMXBUG or SWTBUG) should readily enable you to locate XBASIC's program-buffer area where your program has been stored. After this, it will only be necessary to clear this particular area before writing another test program. Cleaning-up is to be highly recommended, otherwise you'll end up being confused by garbage left over from a previous trial.

TOKEN HASHING & TABLES

Instructions to XBASIC are given either by means of COMMANDs or STATEMENTs. Commands are those instructions which will execute only when the READY prompt is being displayed, such as LIST, RUN, SAVE or LOAD, but will not execute if they appear in a program-line. Statements, such as PRINT, INPUT, FOR or NEXT, on the other hand, are capable of executing directly OR as part of a program-line. Be that as it may, one of the first things that XBASIC does when you hit C/R is to "parse" the line - that is, it examines it for correct structure, or syntax - and will not accept the line if it contains a major error, such as a missing quote or a missing parenthesis. At the same time it converts instructions to "tokens" in a Work-Buffer distinct from the actual Input-Buffer, and classifies and codes the different types of Variables, as we saw previously.

In Immediate-Mode, the parsed line will then be immediately executed, but in Program-Mode it will be transferred from the Work=buffer to the Source-Buffer, and integrated into the complete program by adjustment of the linkage-pointers at the start of the appropriate lines.

A token is nothing more than a code-number for each instruction. We've already come across the example of '69' being the token for '='. Others are GOTO 01, GOSUB 02, RESUME 03, and so on. And what advantage does tokenising serve?, you ask. The main benefit is speed of execution of your program. Without tokens, let's assume that XBASIC meets the word PRINT in a program-line. In such a case it would have to scan an entire Statement-Table until it identified the word PRINT, at which point would be stored the address of the subroutine which actually tells XBASIC how to correctly interpret and execute this instruction. With tokens, however, all occurrences of the word PRINT have been replaced by its token 07 during the parsing stage, so now, when XBASIC comes across 07, it "knows" that the 7th location in a Statement Vector-Table contains the address of the PRINT subroutine, and BINGO, it can go straight to work without having to carry out an exhaustive table-search each and every time.

We have already seen that it does a similar thing with all its variables, by stacking them in a table, and maintaining pointers to their locations in that table. This is why XBASIC is like a speeding bullet compared to some other BASICs.

On the other hand, XBASIC goes to some lengths to hide its innermost secrets from prying eyes by encrypting or encoding the key words PRINT, INPUT, etc into a coded form, so that an examination of the machine-language coding of XBASIC will not readily show where the Command-Table and Statement-Table are located. Thus the ASCII codes for the individual letters of the word 'SAVE' (53,41,56,45) are first 'encoded' into the codes 8C,68,92,70, and then the command-table is scanned for the encoded code-sequence in order to identify its corresponding token. In addition, of course, the reverse process has to occur when, for example, a line is LISTed to the screen. First, the table is scanned for the token, from which XBASIC is able to find the corresponding secret-code sequence and decode it into its human-readable form before finally shooting it out to the screen. For those who are interested, the encoding procedure involves doubling the ASCII code for a particular letter and subtracting $1A from the result. Thus 'E' (ASCII 45) becomes 8A by doubling, and 70 by subtracting 1A.

Obviously then, if we went through these tables and re-wrote the Commands and Statements in normal ASCII we could save XBASIC the time and trouble of encoding and decoding these encoded forms. In addition, it goes without saying, we would also have to NOP out the encoding and decoding routines. To NOP something out means to replace it with NOP (No-operation) codes, 01 for 6800 machines, or 12 for 6809 computers.

MODIFYING XBASIC

Let's try that! Now that we know that SAVE encodes into the code-sequence 8C,68,92,70, let's scan memory for that sequence. Having found it (it's at location 0682 in one version of FLEX2 XBASIC), we proceed through the Command-Table, decoding as we go, until we come to command SCALE, whose token is 0E, which ends the table. Along the way we were somewhat surprised to encounter the command T[^]@ with token '0D'. Strange, there's no mention of it in the XBASIC article. To satisfy your curiosity as to its function you should enter it in response to the READY prompt. How about doing it right now?

The best way to decode the tables is to initially prepare a decoding table composed of the 26 letters of the alphabet and their corresponding encryption-codes and matching ASCII codes. This will easily allow you to identify the various instructions as you come across them in the Tables, and to convert them to straight ASCII.

The next step is to locate and decode the Statement-Table in the same manner. Note that you cannot stop part way through - both tables, Command and Statement, **must** be done at the same time! The first Statement (token 01) is GOTO, which encodes into the sequence 74,84,8E,84, and which is located at address 0FF3 in the old FLEX2 version. Here you will decode all the way through the Statement-Table, which runs straight into the Function-Table (commencing with FN - token 34), and on into the Operator-Table which encodes the three logical-operators NOT, AND and OR, ending at token 60. Beyond this point will be found several other symbols, such as <, >, =, (, ?, etc., which TSC didn't bother to encode. Note, by the way, that the token for '?' is the same as that for PRINT, which explains why XBASIC accepts an entry of '?' as being equivalent to an entry of 'PRINT'.

Finally, we must locate and NOP out the encoding and decoding sections in order to neutralise their effects. The encoding section can be identified by the code :

```
48     ASLA     Multiply by 2
80 1A  SUBA #$1A Subtract $1A
```

and the decoding section by its inverse :

```
8B 1A  ADDA #$1A Add $1A
44     LSRA     Divide by 2
```

Replacing these 6 bytes with the appropriate NOP code will complete the whole operation.

I should point out here that XBASIC will not actually EXECUTE a program any faster as a result of the above conversion, as all statement-lines will have been tokenised by RUN time. Only operations involving converting lines to a tokenised form (or conversely de-tokenising them back to their original form) will be speeded-up. This includes operations such as LIST, LOAD or SAVE.

A SUGGESTION

I like to standardize commands in the various programs in my library. That is to say that if I wish to exit to my MONITOR program, I don't want to be left wondering whether I should enter MON, or maybe EXIT, or should it be QUIT or even PATCH? So, while I was about this remodelling business in my own system, I decided to standardise on the command 'MON' to return to MONITOR. Makes sense, and in this connection I would like to suggest to program-writers that they develop a set of standard commands, such as MON to exit to MONITOR, FLEX to exit to FLEX, and so on. Of course, by changing XBASIC's 'EXIT' to 'MON' we'll have a byte left over, which we'll bubble-down to the bottom of the table during our conversion, and leave it there as an extra NUL (code 00). Don't forget, too, to go through your XBASIC article and change all references to 'EXIT' to 'MON', should you also decide to go this route.

DISPLAYING XBASIC ERROR MESSAGES

A modification to 6809 XBASIC appeared in a long-ago issue of 68 Micro Journal, which I think bears repeating here, if only for the benefit of newcomers to this excellent magazine. This mod gets rid of those aggravating "ERROR #73 AT LINE XXXX" type of error-messages, and replaces them with the much more readable "ILLEGAL EXPRESSION AT LINE XXXX". In order for this to work, though, you'll need the expanded ERRORS.SYS on Service Disk #19, or key it all in yourself. First of all, for 6800 owners, you should locate the following code somewhere near address $0BC3, while 6809 users will go to the approximate address $0B77, and make the specified changes :

6800 USERS		6809 USERS		
Present XBASIC	Change to	Present BASIC	Present XBASIC	Change to
CE 0C0F	B7 AC20	30 8D 00 xx	30 8D 00 xx	B7 CC 20
BD 05B4	8C AC1F	17 F9 xx	17 F9 xx	8E CC 1F
CE 0122	BD AD3F	30 C8 xx	30 C8 xx	BD CD 3F
5F	5F	5F	17 xx xx	BD CD 24
F7 0126	F7 0126	17 xx xx		12 (12)
BD 0FC6	BD AD24	where xx is some code		

The extra 12 (NOP) will be necessary only for BASIC, not for XBASIC.

A PROGRAMMING TIP

I think every reader knows that PRINT Statements can be chained thus :

```
50 PRINT "Your name is ";N$;" and your age is";A%; etc
```

but how many, I wonder, know that INPUT Statements can also be chained, and even have a final text-string displayed? Thus :

```
50 INPUT "What is your name ",N$; "Your age ",A%; "Thank you!"
```

This is equivalent to the following :

```
50 INPUT "What is your name ",N$
51 INPUT "Your age ",A%
52 PRINT "Thank you!"
```

I'd never seen this mentioned anywhere (other than in a BASIC-PLUS article by Digital Equipment Corporation), so out of curiosity decided to try it with XBASIC, and was quite pleased to find that it worked. Actually, after making due allowance for references to mag-tapes or paper-tape, **the DEC article could almost have been written for XBASIC users.**

XBASIC'S "STOP", "RND" & "LIST"

XBASIC's Instruction-Manual doesn't describe these three instructions correctly. For instance, when a program encounters the STOP instruction, and is then restarted with the CONT command, it resumes operation AT THE LINE FOLLOWING THE 'STOP', and NOT at the Statement following the 'STOP' on the same line (if one exists). If any Statements DO follow the 'STOP', they are thus ignored, so, if you are using STOP as a debugging tool, this should be kept in mind! Depending on the nature of the bug, it may sometimes be advantageous to insert a STOP inside a program line, and at other times at the end of a line. Usually an extra debugging line is inserted at an appropriate point in the program, such as :

```
51 PRINT I%, J$: STOP
```

which will cause the current values of the named variables to be displayed, followed by the STOP message. At this point the value of any variable may be changed, and the program will still CONTinue. But not if the program itself is changed in any way, let's say by adding or deleting a line.

RND(X) generates a number in the range 0 through 0.999999....., and NOT 0 through 1 as stated, so the formula given is valid only for the generation of FRACTIONAL numbers in the range MS up to, BUT NOT INCLUDING, ML. If you wish to generate random INTEGERs over a desired range, the formula should be amended to read :

```
X = INT((ML - MS 3+ 31) * RND(0) +MS)
```

otherwise it will not generate ML itself.

In addition, the description of LIST is incomplete in that it informs us only that LIST may be terminated with a ^C (BREAK), with no mention of the more normal use of ESC to halt or re-start the LISTing, and C/R to abort it while in the halted state.

While on the subject of random-numbers, I've always found it a good idea to incorporate a known random-seed in the RND-generator, by inserting something like the following near the start of any XBASIC program (particularly a Game) which uses RND(X) :

```
50 INPUT "Please enter a number between 0 and 99999 (0 allows the
   computer to choose) ",X : X=RND(-X)
```

This technique is invaluable if the program has a bug, as it is possible to keep repeating the self-same game-sequence (unless, of course, a response of 0 was made). It can also be used in the event of a really hard-fought game, where you just MARGINALLY lost to the computer, and would like to try it all over again using slightly different tactics, to see what the new outcome would be like. Decimal numbers are equally acceptable as a response, eg 123.456,

ANOTHER PROGRAMMING TIP

Talking about Games .... as you well know, lots of games include rules or instructions for play, and usually ask the Player whether he needs those instructions. They take up a lot of room in memory, and if they occur in the wrong place in a program they help to slow down its execution. Besides which, after a while you don't really need them any more, and yet you don't want to throw them out in case a new Player comes along who's not familiar with the rules of play.

In such a case, it's a good idea to remove all these instructions by judicious use of the SPLIT utility. After splitting them off, the EDITOR, or other word-procssing tool, can be used to remove all items specific to XBASIC, such as Line-Numbers, PRINT statements, etc., leaving only the text which was originally enclosed in quotes. The quotes themselves should be deleted, and the file finally renamed to <game.DOC> or <game.INF>. Then, in the actual game itself, a sequence such as the following can be inserted near the start :

```
50 INPUT "Do you want instructions (Y or N) ",Q$
60 IF Q$="Y" OR Q$="y" THEN EXEC,"TTYSET PS=Y: LIST game.DOC: TTYSET PS=N"
70 rest of program
```

A response of 'Y' then causes the program to turn on the PAUSE feature so the LISTing doesn't scroll madly off the screen, but waits for the reader to scan the page before hitting ESC to scroll to the next. Finally PAUSE is turned off, and the program picks up at Line 70. Any response, other than "Y" or "y" bypasses this sequence and proceeds straight to Line 70.

COMPILING SOURCE FILES DIRECTLY FROM XBASIC

If you really want to speed up your XBASIC programs, however, you should make more use of its COMPILE command, by first saving your program in the regular manner and then executing COMPILE "file-name". The reason you need a regularly SAVEd version as well is that a COMPILED program cannot be edited directly. All you can do is to RUN it. It makes for a much more compact file on disk, takes up a lot less room in memory, and also loads much faster! It will have the extension .BAC (for BAsic Compiled), and can be fired up from FLEX with the command 'XBASIC file-name', or from XBASIC itself with the command RUN "file-name". In both cases the program LOADs and RUNs automatically.

Transferring instructions to a .DOC or .INF file, and/or COMPILing, will very often allow you to run those REALLY large programs which won't quite fit into memory otherwise, and is preferable to stripping out those essential REMs from your programs.

But have you ever tried to LIST a .BAC file which XBASIC has in memory? It will snap right back with the message "SOURCE NOT PRESENT", which we know isn't true, because it's perfectly capable of executing the program in response to RUN. Would you like to be able to LIST it?

Here's how! First load XBASIC, then execute MON (or EXIT if you chose not to standardise). Now, somewhere in the vicinity of address $099B for 6800 owners, look for code 7D 011C 27 05 86 40. The important byte is the 7D, which should be changed to 39. 6809 users will look in the vicinity of address $0900 for the code sequence 6D C8 57 27 xx 86 40, and similarly change the 6D to 39. Then execute a jump to XBASIC's Warm-Start with GMXBUG's J 3, or its equivalent in your Monitor. Now LOAD your .BAC program with 'RUN "file-name"', and wait for it to halt with a request for operator-INPUT. When this occurs hit BREAK (^C) to interrupt the program, or if you're really impatient hit BREAK as soon as your program is LOADed and actually RUNning, and then LIST. Did you ever see such a compact program format? Makes you wonder how the program could possibly RUN with so much essential information apparently missing! No wonder it takes up so much less room in memory, or on disk. Now you see why you can't change it if it has a bug in it. You must identify the buggy line, go back to your originally saved .BAS file, debug this version, SAVE it once more, and finally COMPILE a new .BAC file.

To Be Continued Next Month

Serial Communication Links Intelligent Microcomputers

MOTOROLA INC.

Robert A. King
Manager Technical Communications
Semiconductor Products Sector, Austin
3501 Ed Bluestein Boulevard
P.O. Box 6000, Austin, Texas 78762

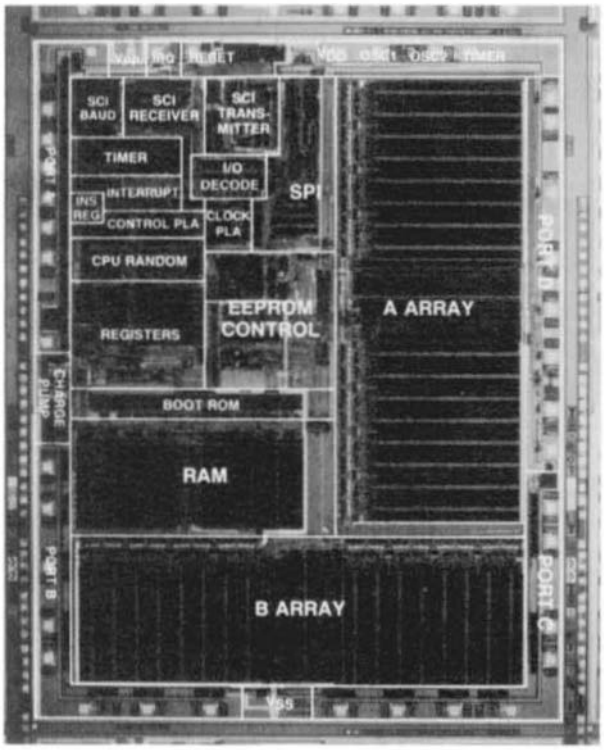
Introduction to Serial Communication

A complete block diagram of this general purpose MCU is illustrated in figure 1. Being implemented in HCMOS technology gives this device the advantages of operating at high speed while consuming very low power. Noise immunity is also a key reason for using HCMOS in this and many other new integrated circuits.

Several discrete functions have been pulled on-chip to lower both system chip count and the overall cost. Features such as the on-chip EEPROM (4160 bytes), bootstrap ROM (240 bytes) and Static RAM (176 bytes), and the 24 bidirectional I/O lines are included on the chip. The I/O lines are grouped to form three 8-bit parrallel ports A, B, and C. The fourth port, D, is a 7-bit fixed input port used to operate and monitor the serial systems. The sophisticated 16-bit timer system adds to the overall flexibility of the MCU. The timer is ideal for accurately tracking events within the system making

An alternative to the traditional parallel bus architecture is the serial network link used to distribute processing. In so doing, much of the buffering and control logic of a standard system is significantly reduced and in some cases eliminated. Many of todays single-chip microcomputers (MCUs) are enhancing system performance by supporting a variety of serial standards. One such MCU produced by Motorola is the MC68HC05C4.

This is an 8-bit device that offers two distinct serial protocols: they are the Serial Peripheral Interface (SPI) and the Serial Communications Interface (SCI). Each type supports specific application areas. The SPI standard is used for local data transfers usually on a single printed circuit board. The SCI on the other hand, can support communications between greater distances. This could include transfers between different boards within a system or from data moves between box systems.



sure that operations happen at the correct time. The CPU portion contains a control section, an ALU, and a microproceesor.

The serial ports of figure 1 are part of port D. Included are the control lines associated with the actual data transfer process. A serial system can be designed with minimal interfacing hardware by using port D.

Linking Multiple MCUs Using SPI

The SPI is a well defined standard that has evolved as the result of pin-limitations on some early MCUs. By using the SPI interface scheme, multiple MCUs have a communications link allowing them to transfer data to and from each other as well as between other external peripheral devices on a printed circuit board (PCB). There are four primary signal lines associated with linking HCUs

with the serial standards. These are the data in, data out, clock, and slave select. The possible configurations supporting this method are a three wire system consisting of one master MCU and a four wire hookup which has both multiple masters and slave MCUs.

In figures 2a and 2b, two possible configuration schemes are shown using multiple MCU devices. The first illustration, figure 2a, has the single master implementation with four slave MCUs. In figure 2b, the required arrangement for a four master and two slave system is displayed.

The slave select is an active low input to the MCU. It is used as the control to the device indicating either master or slave status. A low shows a slave status while a high indicates mastership of the MCU. The slave select line of the master MCU in figure 2a is pulled to V_{DD}, therefore, it is held at a constant high level. The I/O port (port D) is used to choose one slave of four to allow data transfer to only one slave device at a time. This is performed under software control.

In figure 2b, the port D lines choose the appropriate slave to transfer data to or from a master device. Only one of the three masters can control the communication channel at a time. Applications where this type of interconnect would be appropriate include numeric calculating systems not having a lot of time for processing data but where the result is needed quickly.

Write and Read Cycles

To illustarte how the master and slave MCUs transfer data, refer to figure 3. With the slave select of the master high or at 5 volts and the slave MCU's slave select low at 0 volts, the transfer takes place. The SPI clock generator produces a stream of pulses.

During a write cycle, the internal bus does a parallel load of the master's 8-bit shift register. This data byte is then shifted out serially to the slave MCU via the master out/slave in (MOSI) pin. Following the serial transfer of the data to the slave, the read buffers are loaded with the data. Finally, the internal bus receives the original data. The old data that was in the slave device is simultaneously shifted out to the master over the master in/slave out (MISO) pin.

A read cycle operates in a similar fashion to that of the write cycle. In the case of a read cycle, the master MCU recieves data serially into the shift register from the slave by way of the MISO pin. After the register is completely loaded, the read buffers receive the data in a parallel format. The data is then available for use by the internal bus. The software and hardware working together make the master/slave configurations and data transfers possible. External interfacing hardware requirements are minimal while the MCU offers a very efficient instruction set.

Interrupts

An important consideration for successful serial communication is to make provisions for a variety of interrupts. This allows efficient use of the MCU ports and serial bus. There are five types of interrupts available on the MC68HC05C4 MCU. They are the IRQ, SPI, SCI, Timer, and the non-maskable software interrupt (SWI). To execute a software routine from the on-chip memory, basic checks are made to insure proper operation. After reset, either from power on or externally, the interrupt status is continually being watched. Figure 4 is a flowchart showing the steps required for program instruction execution and the interrupt monitoring sequence.

As long as the (I) bit in the condition code register is set, new program instructions are fetched and executed. When this bit is not set, a poll routine is executed to determine which type of interrupt is requesting service. Once the service request has been satisfied, the internal state of the machine is returned to its original values prior to the interrupt. Program execution will continue from this point.

Internal Registers

The Serial Peripheral Control Register (SPCR) contains several bits that control the operation of the serial bus. There is an interrupt bit, an output enable bit, a master bit, clock polarity and phase bits, and two clock rate bits.

A second register, the serial peripheral status register (SPSR), has a flag bit for data transferring, a mode fault flag, and a write collision status bit used to notify the user that during a data transfer, an attempt to write to the data register was made. If a collision is detected, the original transfer is not interrupted and the write attempt to the data register is unsuccessful.

Linking Peripheral Devices

Numerous application areas can be supported using the SPI standard as the link. Off the shelf logic, memory, and other peripheral devices are ideal for the above MCU configurations. Included in this group would be EEPROMs, RAMs, LED drivers, A/D converters, frequency synthesizers, and I/O expansion devices. Further simplifying the design, most of these devices require only an input for data plus a clock. Their one direction data flow characteristic also helps with the task of interfacing.

By using the serial approach, many system control functions can be performed by these intelligent controllers, the MCUs. The requirement for accuracy plus the need for high data transferring is maintained by using the serial bus communications link.

One possibility would provide communication between MCUs and peripheral devices that are on a single board computer (SBC). By reducing parallel bussing on a SBC, much valuable board space can be saved. Status of external functions can easily be monitored. Data acquisition systems that convert analog data to digital signals interfacing with the MCUs on the SBC can control such analog functions as temperature, pressure, or speed. Transducers would be used as part of the data acquisition system.

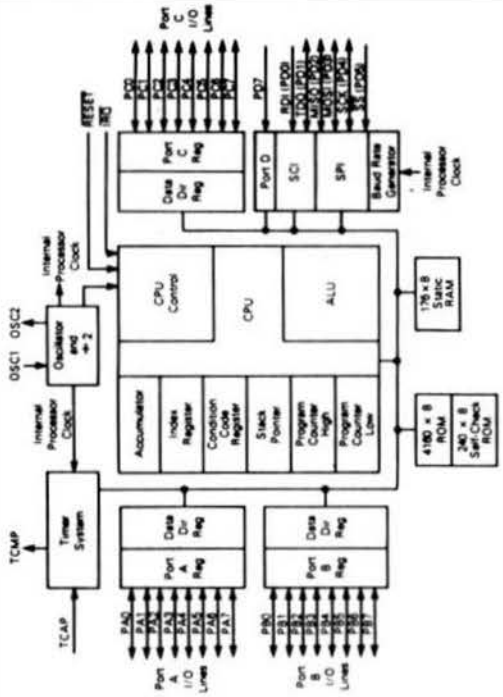


Figure 1

A block diagram of the MC68HC05C4 is shown. Its multiple I/O and serial ports enhance its capability of being designed into many application areas.

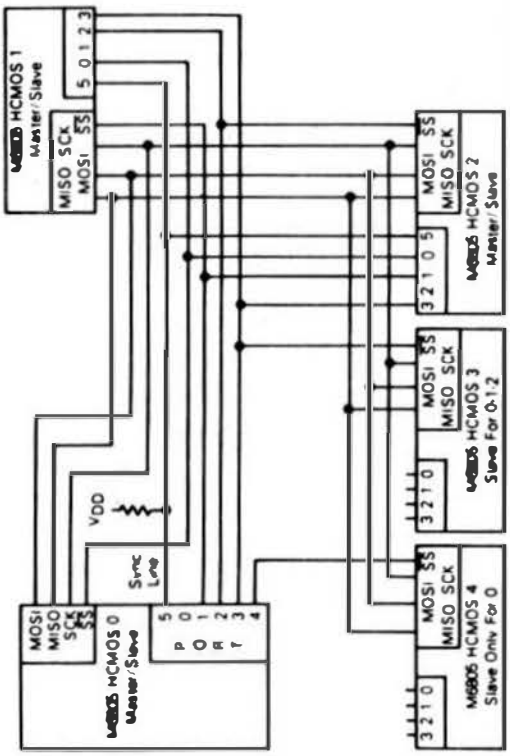


Figure 2a

A single master with multiple slave MCUs is being illustrated. This is only one of several possible configurations.

Summary

The basic MCU system which transfers data serially has many advantages. For local communications, such as between MCUs or peripheral devices on a single PCB, the SPI is ideal for reducing hardware interfacing interconnects and parallel buses. The MC68HC05C4's serial standards are of a general nature, simplifying the basic design while reducing overall system cost. This allows them to be designed into systems with little effort.

Programming these devices is similar to programming a standard M146805 family member. It has 62 instructions in its instruction set. There are four different types of instructions. These are the register/memory, read-modify-write, branch, bit manipulation, and control.

By combining all of these features added to the device flexibility, the MC68HC05C4 is the perfect solution to many application areas. It is easy to design-in, program and offers high reliability for the end system.

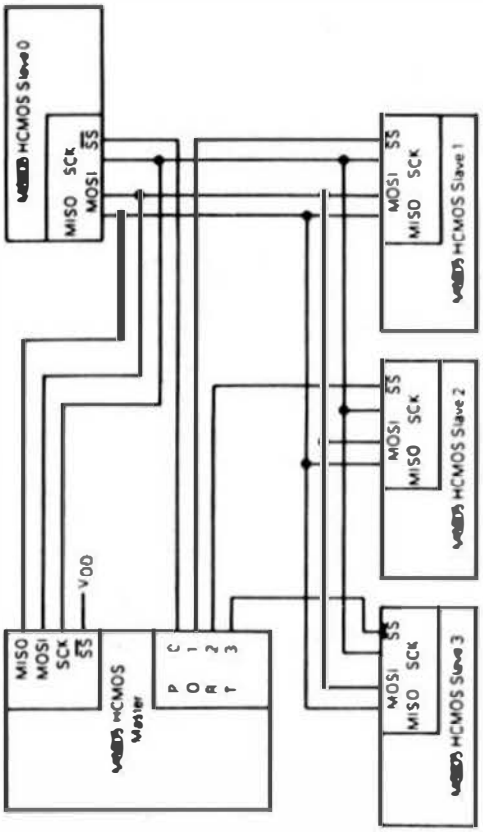


Figure 2b

Multiple masters can be supported using the MC68HC05C4 MCU.

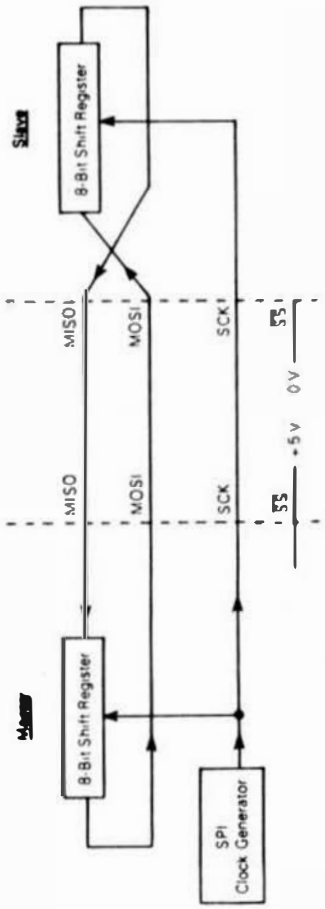


Figure 3

Serial data is shifted through registers by the SPI clock generator. Data is made available in a parallel fashion for the use of the internal bus.

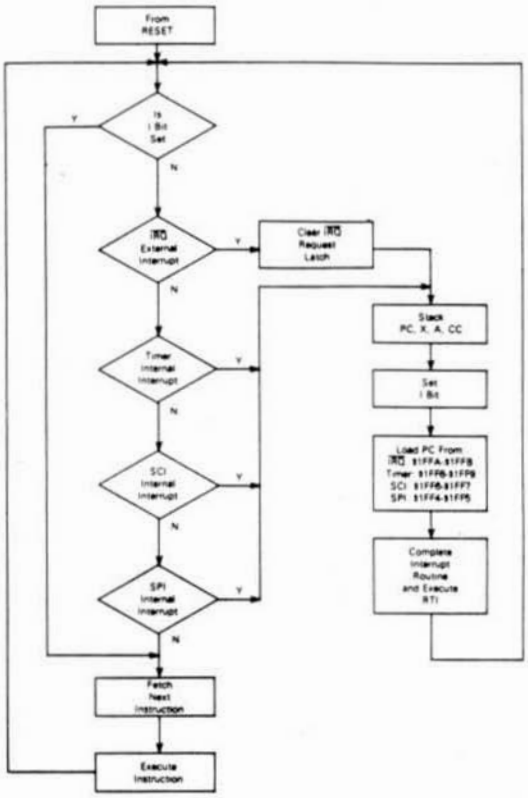


Figure 4

A flowchart of the interrupt process is shown above. Both hardware and software interrupts are checked during program execution.

MOTOROLA INC.

EDITORIAL CONTACT:
Angela Hatfield
(602) 952-3613

READER CONTACT:
Technical Operations
(602) 244-7579

INQUIRY RESPONSE:
P.O. Box 52073
Phoenix, AZ 85072

NOW AVAILABLE FROM MOTOROLA — MC68020 AUDIO CASSETTE COURSE

Phoenix, Arizona, March 4, 1987 ... Motorola Semiconductor Products Sector Technical Operations announces availability of its new MC68020 Audio Cassette Course: An Introduction to the MC68020 32-Bit Microprocessor (MTTA2). The self-paced audio cassette course comes in an attractive new package and contains three tapes (four and one-half hours long), fully illustrated course notes, the MC68020 User's Manual, and related support literature. The course covers the major enhancements of the MC68020 over the original MC68000, including pin and bus operations, addressing modes, instruction set, and exception processing.

The course material is designed in a modular fashion with clearly stated objectives, comprehensive examples, evaluations for each module, and supplemental study references to enable the learner to customize his learning experience. The course is intended to enable a design engineer or programmer already familiar with earlier MC68000 microprocessor family members to successfully use the enhanced MC68020. Motorola's MC68000 (MTTA1) audio cassette course or equivalent experience is a required prerequisite.

Bit-Bucket

By: All of us

Motorola's MC68000 course is being upgraded from the current overview to reflect the same high standards as the new MC68020 audio cassette course, and will be available July 1, 1987. Both audio cassette courses, the MTTA1 (68000) and MTTA2 (68020), highlight the same technical information as the formal instructor-led courses, MTT8 and MTT20, respectively.

The MTTA2 (68020) audio cassette course retails for $95.00, as will the upgraded version of MTTA1 (68000). The current version of MTTA1 (68000) retails for $60.00, but may be upgraded when available for an additional $35.00. A special price of $140.00 is offered to a customer who purchases both the current version of MTTA1 (68000) and the new MTTA2 (68020). Local sales tax should be added to all orders.

To receive a catalog describing all of Motorola Technical Operations' course offerings and to order a copy of the MC68020 audio cassette course, circle the reader service number or call Motorola Technical Operations at 1-800-521-6274.

Motorola Semiconductor Products Sector Technical Operations offers a wide selection of technical courses ranging from courses for non-technical people to courses for systems integrators, from introductory courses on MPU design to advanced courses on the MC68020. Motorola Technical Training courses are scheduled in training centers in the United States, Canada, and Europe, and are also offered in South America and the Asia-Pacific Crescent.

The National Computer Conference
McCormick Place
June 15-18, 1987 • Chicago, Illinois

CONTACT: Roger Halligan

(312) 222-0197

MOTOROLA CHAIRMAN WILL BE

NCC '87 KEYNOTE SPEAKER

Chicago, IL, (April 3, 1987)...Provocative high-tech pioneer Robert W. Galvin, chairman of Motorola, Inc., will share his insights on the many challenges currently facing the computer industry as the keynote speaker at the 1987 National Computer Conference (NCC '87). The conference will be held in Chicago June 15-18.

In addition to directing one of the world's leading high-tech companies, Galvin also has spearheaded Motorola's communications campaign to raise public awareness of the importance of fair, effective world trade. He has spoken out on the problems faced by U.S. business through Motorola's extensive program entitled, "Meeting Japan's Challenge".

Motorola is well known for its M68,000 family of microprocessors. These chips (along with similar microprocessors from other manufacturers) have been responsible for the dramatic change in computing price/performance over the past few years. By using these powerful microprocessors, manufacturers have put the power of a mainframe computer into desktop systems.

Galvin, 64, began working for Motorola in 1940 and joined the company full time in 1944. In 1956, he was named president. In 1964 he became chairman and chief executive officer. In 1968, he relinquished the title of CEO, but remained chairman of the board, Motorola's senior officership. Galvin, a Marshfield, Wisconsin native, attended the University of Notre Dame and the University of Chicago. He also holds honorary degrees from several other highly-regarded academic institutions. He is a past president of the Electronic Industries Association, a director of Junior Achievement of Chicago and chairman of the Board of Trustees of the Illinois Institute of Technology.

Based in Schaumburg, Illinois, Motorola is one of the world's leading manufacturers of electronic equipment, systems and components produced for both the United States and international markets. As a leader in the high-technology markets, Motorola is one of the few end-equipment manufacturers that can draw on expertise in both semiconductor technology and government electronics.

Ranked among the United States' 100 largest industrial companies, Motorola has some 95,000 employees working in major facilities in 18 countries.

The National Computer Conference is sponsored by American Federation of Information Processing Societies, Association for Computing Machinery, Data Processing Management Association, IEEE-Computer Society and Society for Computer Simulation.

For information on attending NCC '87 call 800-NCC-1987 or 800-225-3333; or write, NCC '87, AFIPS, 1899 Preston White Drive, Reston, VA 22091; or Telex—TWX 710-833-9037. For information on exhibiting call 800-222-5755 (in Virginia, 703-620-6918).

Classifieds
As submitted - No Guarantees

DAISY WHEEL PRINTERS
Qume Sprint 9 - $900 Qume Sprint 5 - $800

HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.

3 - Dual 8" drive enclosure with power supply. New in box. $125 each.

5 - Siemens 8" Disk Drive , $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, $495.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $250 ready to run.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Paralell Port, MP-09CPU Card - $900 complete.

(615) 842-4600 M-F 9AM to 5PM EST

•••

Wanted to buy. SWTP MP09 CUP CARD, SS50C 64K or 256K memory. Details to Robert Steedman.
RMB 9010
Omeo Highway Sarsfield
3883, Australlia

•••

Wanted Percom LFD400 floppy disk controller card.Contact: Jack Bowodin (219) 262-2126

The Macintosh™ Section

Reserved as a

A place for your thoughts

And ours.....

Mac-Watch

THE STRIPPER

Nope we havn't gotten into burlesque or movie reviews. Not yet, anyway. What we have here is a review of a new and somewhat novel accessory for the Mascintosh - the ***Cauzin Softstrip System.***

Now, many of you oldtimers wil remember a similar system promoted by Byte magazine, some years back. Although not the same, both the Cauzin System and the Byte effort were of the same family - reading of printed strips from paper. The Byte system read bar code type strips. Not too efficient, even compared to the Cauzin system. Also, the Byte system left it up to the reader-user to solve the hardware end. Eventually, it was probably the hardware (or lack of same) that caused that effort to slowly fade away. Byte, because of their size and clout could have made it go, but the hardware availability thing was never resolved. With the Cauzin System it is.

There are other significant differences between these two. First, the Cauzin system is far more efficient in storing data to a piece of paper. An 8.5 by 11 sheet of paper (about the size of the average magazine page) could hold about one or two Kbytes of data, bar-code style. Slightly more, according how tightly the bar-code strips were 'pasted' to the page. Bar-code microcomputer data use never really got far off the ground. And there is now few who remember that effort. So, we will not dwell there any longer.

The Cauzin System is a better thought out system of strip storage.The data printed by the Cauzin System is in a format known as *dibits*. Each zero (binary) dibit is two successive squares with a black followed by a white. A one is a white followed by a black. So, each byte of data is composed of 16 black and white squares in a row.

The Cauzin System is composed of two major items, the reader and it's software. Available now for both the Mac and IBM crowd. The rest will have to do without until some enterprising hacker decides to implement the strip software for other systems, or Cauzin gets around to it.

The storage capacity of the Cauzin System is far more efficient than the bar-code way. Using a dot matrix printer - for the Mac, the ImageWrite, density can be about 700-800 bytes per 9 inch strip. According to how tightly they are 'pasted' to the page that will yield about 3 to 4 Kbytes of data per page. The LaserWriter produced strips are able to lay-down about 3 to 4 Kbytes per strip, or about a full page of ImageWriter strips per page. Also, Cauzin offers to publications using their strips a rate for high density negatives (you send the data to them, they send the negatives back, for a small fee); this density packs about 5 to 6 Kbytes per page.

The Reader

The reader is a device that is placed over the strip to be read. It has an internal track and a small motor that drags an optical reader over the strip. The time to read an average 9.5 inch strip is about 30 seconds. In that period of time it has read each line several times (to insure accuracy) and downloaded the data to the Mac in 4800 baud burst. We read thousands of bytes and several dozen strips witout error. It is sturdily constructed and very reasonably priced at $195.00, with reading software.

If you desire to make your own strips then you will have to purchase the STRIPWARE™ Stripper™, for a bite of about $19.95. Although not a very heavy price, but due to the essential part it plays in the entire scheme, and that Cauzin certainly must be doing what they can to insure the spread (and survival) of Cauzin technology, I would have thought that it would have been a part of the original package. ***Everybody should be making and distributing strips!***

Another surprising aspect relating to the above is the reproduction for distribution of strip software you might develop. *You must pay Cauzin a fee for each one sold commercially!* Personally I feel that that is a real hummer. Seems there just may be too many barbs on the wire that separates success from failure. Especially when you consider the fate of bar-code data schemes and the fickle nature of the Mac market. I called Cauzin about this some weeks prior to this writing. They indicated that was under review, and someone with better information, on this subject, would get back to me the following Monday. They never called. So, I only know what the book says!

As of this date I find little strip software advertised in any of the Mac directed magazines (first week of October). Fact is, the most we have seen, are in Cauzins ads. And even there, in addition to several small programs, you find that many (most) of the strips are advertising hype, instructions for other software that most users do not have, or some other useless data. Most users I discussed this with expressed the feeling that they get more indepth reviews and advertising in the magazines. They would like to read more useful stuff with their readers. *No one wants to shell out about 200 bucks to read hype about things they don't have!*

Software - good software - is going to be the heart and lungs that keeps this technology going. Software restrictions will certainly not help. Unless more and better software appears in strip form, at a reasonable price, I fear that this will be but another passing fad.

Also, several of the larger software vendors made remarks to the effect that because of the low price of bulk diskettes (also no fees paid for anything on the disk, to the disk manufacturer) and that the storage density of diskettes is so much higher, diskettes seem the natural basis for media distribution. All Mac & IBM users have disk drives.

Also during discussions of this type of media with several software vendors, all, I repeat all, told me that they are not willing to make a commitment to this until they see some movement towards stability. That movement will come from magazine readers and users. Until then a big ? hangs over this new segment of data collection, storage and distribution.

The following have indicated, to some degree or another, that they will use this strip system somewhere within their operation. *Bar Code News, Byte, Data Based Advisor, Keyboards, Computer & software, Library High Tech News, MacUser, McGraw-Hill, Nibble, NibbleMac, PC World, Pergamon Press, Computing, Village Voice and several Users Groups.*

For our Mac readers, we would be interested in running strips in our Mac columns, if you wanted them. So, you Mac readers and users let me know.

Conclusion

Frankly, I liked the thing. It worked every time. Slow but accurate. From a publishing standpoint it could fill a void - that of compressing the program listing space. Or even better, publishing programs where none were published before. However, unless a lot of readers have the reading device, forget it. And I don't think a lot of users will pop for the system until they can see some real benefit, to them, for the cash outlay. Which just goes to show how difficult it is to get a new technology off its knees and into the running position. Even a really good one. And the Cauzin System is a good one, from a reading standpoint. From a software developers angle, well, I don't know...time will tell.

Additional information can be had by contacting:

CAUZIN Systems, Inc.
835 South Main St
Waterbury, CT 06706
(203) 573-0150

EOF

*Editor's Note: Presently there are several systems able to run the Cauzin Strip protocol (IBM PC, Macintosh and Apple II). As it is machine or operating system dependent, it will need to be ported to each newcomer. This normally leaves some of the smaller populated systems never receiving a port. The OS-9 and FLEX/SK*DOS community being one. For that reason we would need someone to do a port of the software if we were to use strips for code..both ASCII and binary.*

Of course, our Macintosh readers will have no problem, but the rest will have, unless we get a port. I just wondered if any of you would like to make an attempt. If so, let me know and I will take it from there.

We would be willing to publish software (code/binary) in 68 Micro Journal, if there were sufficient numbers of you out there with Cauzin readers.

DMW

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUCCESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uees; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINTC3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINTC4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included in ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

Ωmega . . . The Desktop Mainframe

* 12.5 MHz MC68020 32-bit microprocessor & MC68881 FPCP as standard.
* 1 megabyte of non-volatile, zero wait-state STATIC RAM and 128k/256k bytes ROM.
* 25.5 megabyte (unformatted) Winchester Hard disc as standard.
* 1.0 megabyte (unformatted) DS/DD 80 track floppy disc drive (specify 5¼" or 3½").
* Five RS-232C serial ports (one may be configured as an RS-485 network node).
* One Parallel printer port and one 16-bit bi-directional parallel port.
* Non-volatile clock calendar and power supply supervisor.
* OS-9/68K™ 'Professional' with full MC68881 FPCP support and 'C' compiler.

OMEGA/12.5/OS	OMEGA w/12.5 MHz 020/881 & OS-9/68K™	$4750.00
OMEGA/16.6/OS	OMEGA w/16.67 MHz 020/881 & OS-9/68K™	$4950.00
OMEGA/1MB	1 megabyte additional STATIC RAM	$ 895.00
OMEGA/SER9	9 port RS-232C serial expansion board	$ 645.00
OMEGA/GRF	640 x 480 x 4 bits/pixel ACRTC graphics I/F	$ 925.00

PRICES INDICATED INCLUDE SHIPPING AND APPLY TO U.S. CUSTOMERS ONLY

VISA/MASTER CARD ORDERS ACCEPTED

OS-9 is a trademark of Microware Systems Corporation

NON-U.S. CUSTOMERS SHOULD CONTACT THE NEAREST DEALER FOR PRICE & DELIVERY INFORMATION

NORTH & SOUTH AMERICA

LLOYD I/O INCORPORATED
19535 NE GLISAN
P.O. Box 30945
PORTLAND, OR 97230 (USA)
TEL: (503) 666-1097
TLX: 9103805448 LLOYD I O

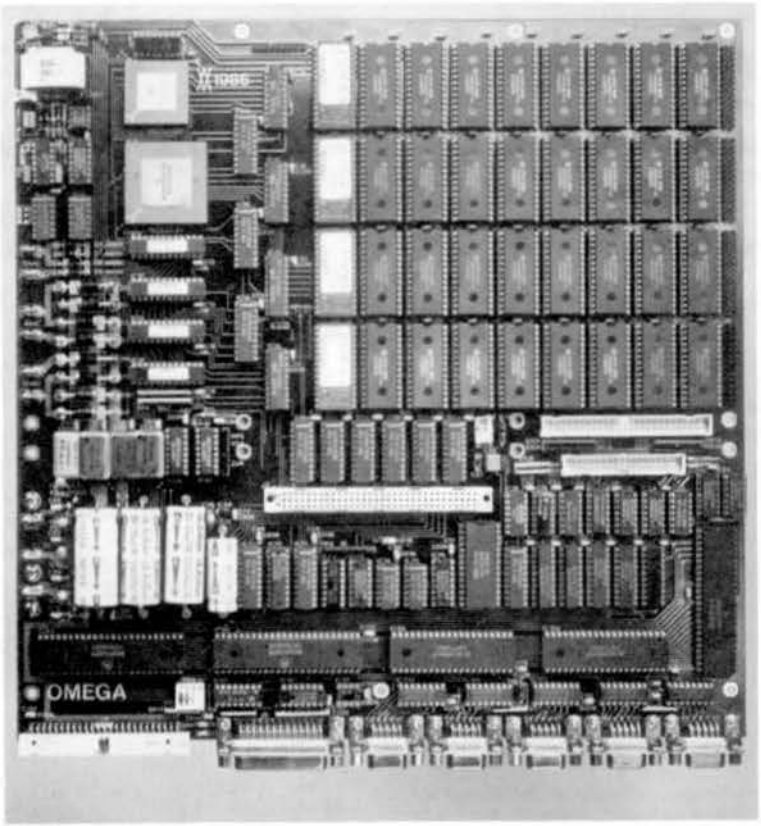
ALL OTHER ENQUIRIES

WINDRUSH MICRO SYSTEMS
WORSTEAD LABS.
N. WALSHAM, NORFOLK
NR28 9SA, ENGLAND
TEL: (0692) 404086
TLX: 975548 WMICRO-G

CONTINENTAL EUROPE

SNIJDER MICRO SYSTEMS
SCHOOTEINDSEWEG 8a
5756 BD Vlierden
The NETHERLANDS

TEL: (0)4930-11975-13666



12" x 12"

On-board power supply . . . only requires external 10 VA transformer.

5 x RS-232C serial ports.

SCSI initiator.

FDC controller

Parallel printer port.

16-bit bi-directional I/F

Non-volatile clock calendar

Fully buffered bus expansion

QUANTITY DISCOUNTS
2-5 less 5%, 6-9 less 10%
10-24 less 15%, 25-99 less 20%
100+ less 25%

12.5 MHz MC68020 32-bit processor and 12.5 MHz MC68881 Floating Point Coprocessor plus 1 Mb non-volatile, zero wait-state 100% STATIC RAM all AS STANDARD!

Ωmega . . . The OEM's CHOICE

OMEGA/12.5	OMEGA w/12.5 MHz 020/881 & 1 MB RAM	$2750.00
OMEGA/16.6	OMEGA w/16.67 MHz 020/881 & 1 MB RAM	$3295.00
OMEGA/LTX	OMEGA 120/240 VAC Line Transformer	$ 22.00
OMEGA/1MB	1 megabyte additional STATIC RAM	$ 895.00
OMEGA/SER9	9 port RS-232C serial expansion board	$ 645.00
OMEGA/GRF	640 x 480 x 4 bits/pixel ACRTC graphics I/F	$ 925.00
OMEGA/OS9	OS-9/68K™ 'Professional' configured for OMEGA	$ 595.00

PRICES INDICATED INCLUDE SHIPPING AND APPLY TO U.S. CUSTOMERS ONLY

VISA/MASTER CARD ORDERS ACCEPTED

OS-9 is a trademark of Microware Systems Corporation

NON-US. CUSTOMERS SHOULD CONTACT THE NEAREST DEALER FOR PRICE & DELIVERY INFORMATION

NORTH & SOUTH AMERICA

LLOYD I/O INCORPORATED
19535 NE GLISAN
P.O. Box 30945
PORTLAND, OR 97230 (USA)
TEL: (503) 666-1097
TLX: 9103805448 LLOYD I O

ALL OTHER ENQUIRIES

WINDRUSH MICRO SYSTEMS
WORSTEAD LABS.
N. WALSHAM, NORFOLK
NR28 9SA, ENGLAND
TEL: (0692) 404086
TLX: 975548 WMICRO-G

CONTINENTAL EUROPE

SNIJDER MICRO SYSTEMS
SCHOOTEINDSEWEG 8a
5756 BD Vlierden
The NETHERLANDS

TEL: (0)4930-11975-13666

LLOYD I/O

TM INC.

ESTABLISHED 1979

ASSEMBLY LANGUAGE DEVELOPMENT SYSTEMS

VANTAGE™ is your secret weapon that speeds the design-code-test phases of developing your ideas into real concrete products. **VANTAGE** is a three part package of integrated software including:

ED™ ..... Editor
CRASMB™ ..... Macro Cross-Assembler
CRACKER™ ..... Symbolic Debugger

The system speeds up development time because the editor, assembler, and debugger talk to each other directly with out having to exchange data on slow disk files. In fact the only time you need to use the disk is to save your source code when you go home for the day.

VANTAGE is compatible with EPROM programmers available on some OS9/68000 computers. EPROM programming is made easy with the built in program, verify, check, copy, and read EPROM commands.

The *fast* full screen **ED**itor is used to code your assembly language program. It is very powerful, including line numbers, source code cross-referencing, 10 edit buffers, command macros, side file functions, access to OS9, . . . etc. When you are ready to assemble, you simply call the cross assembler. It assembles the source code right out of the editor's text buffer(s).

CRASMB is packed with many features to speed your development job. Powerful macro features, symbol cross referencing by line numbers, and plain English error messages are among these. **CRASMB** offers the best in compatibility with the manufactures' assembly languages. The generated object code is placed directly into the debugger's 64K of target ram, or on a disk file in S1, Intel Hex, or Binary forms. The symbol table remains intact for use by the debugger.

CRACKER is chalk-full of so many features that it allows you to attack a debugging situation several ways. Since you have just assembled your program, your object code is already in memory.

CRACKER features a software simulator, dis-assembler, interactive assembler, memory examine, memory search, dynamic breakpoints and other supportive commands. Breakpoints are the key in testing software, so we went all out in our research and development of the breakpoint system. We looked at other debuggers on several different computers, and talked with many assembly language programmers. **CRACKER** has easy to use breakpoints, but at the same time they are very powerful, including; conditional breakpoints, unlimited scope, memory protection, many types of actions including the execution of a macro-breakpoint-language for simulation of hardware devices. Expressions may use symbols that were defined in the assembly process to specify memory address values. Breakpoints remember the symbol names making it easy to make changes to your source code without having to kill and enter new breakpoints. The simulator allows you to execute your object code in a controlled environment. If things get out-of-wack, your memory protection and breakpoints will keep the simulation under control. Single stepping displays the registers and either a dis-assembly of the instruction about to be executed or a listing of the source code line from the editor that generated the object code, i.e. source code debugging is built in.

SERVICE includes a 1 year free update period from the time of purchase. We ship within 4 working days and use *UPS/BLUE*. Each copy is custom manufactured to your computer disk size and format.

Supported CPU's

CRASMB	**CRACKER**
6800,02,08	6800,02,08
6801,6301	6801,6301
6303	6303
6804	6804
6805	6805
6809	6809
6811,H11,HC11	6811,H11,HC11
6502	BY 2Q87
1802	BY 2Q87
TMS7000,20,40	BY 2Q87
8041,8741	BY 3Q87
8048,20,21,22,ETC.	BY 3Q87
8051,31	BY 3Q87
8080,85	BY 3Q87
Z8	BY 3Q87
Z80	BY 4Q87

For OS9/68000 ..... $795

Includes all Supported CPUs

(512K Memory Required)

CRASMB ONLY (OS9/68000) $432
CRASMB ONLY (OS9/6809) $399
ED ONLY (OS9/68000) ..... $ 63
CRACKER and **ED** ONLY (OS9/68000) $495

DEALER TELEPHONES

USA	
SE S.E. Media Supply	1 615 842 4600
NE Frank Hogg Lab, Inc.	1 315 474 7856
SW GESPAC, Inc.	1 602 962 5559
NE Thomas Instrumentation	1 609 987 4280
NW D & S Industries	1 206 827 7300
NW Stylo Software	1 208 529 3210
AUSTRALIA	
Paris Radio Electronics	0 2 344 9111
DENMARK	
H.C. Andersen Computer A/S	0 1 52 44 04
ENGLAND	
Vivaway Ltd.	0 582 423 425
Windrush Micro Systems	0 892 404 086
FRANCE	
Microdata-Soft	1 768 8080
Computer Dialysis France	1 4789 8442
WEST GERMANY	
Dr. Rudolf Keil GmbH	0 62 03 6741
Zacher Kleincomputer	0 85 25 299
JAPAN	
Seikou Electronics Co.	03 832 6000
Microboards, Inc.	0 474 22 1741
SWEDEN	
Micromaster Scandinavian AG	018 138595
SWITZERLAND	
Elsoft AG	056 83 33 77

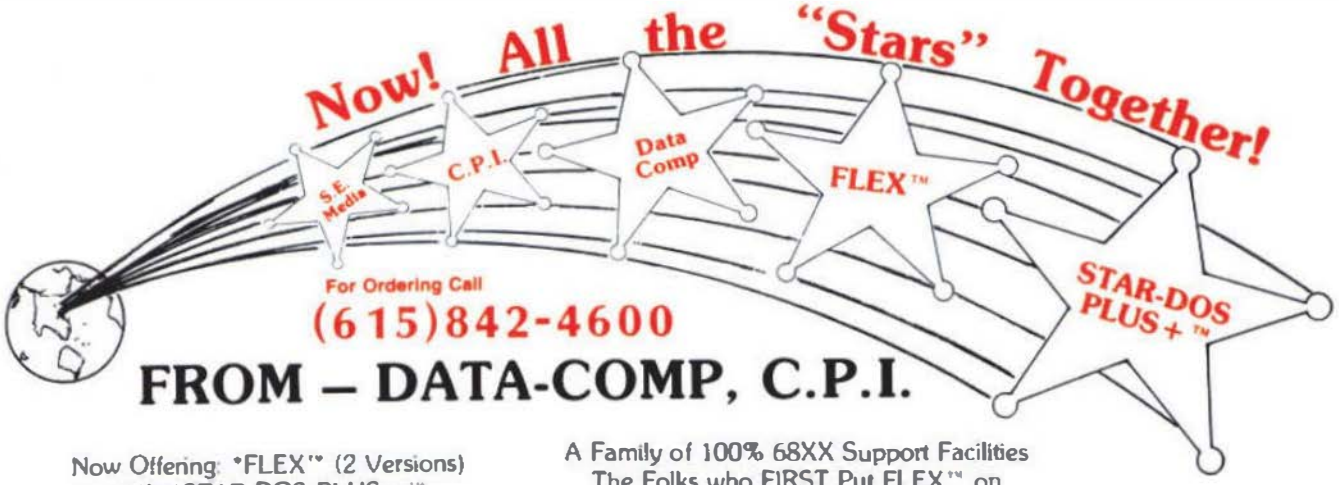
VISA – MASTERCARD

DEALER and OEM inquiries invited

VANTAGE, ED, CRASMB, CRACKER are trademarks of LLOYD I/O, INC.

OS9 is a trademark of Microware
Our Development System is a GMX Computer

LLOYD I/O, INC. • P.O. Box 30945 • Portland, OR 97230 • USA
Attn.: Frank Hoffman • Phone 1-503-666-1097 • Telex 910 380 5448 LLOYD IO



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO 11	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing

S - 50 BUS / 68XX

Board and/or Computer
Terminals-CRTs-Printers
Disk Drives-etc.

REPAIRS

This

Not This

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to **'ALL'** S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. *Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and *confirm a Data-Comp service & shipping number*! This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but **NO** advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates *must be requested*. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with the item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4607
Telex 5106006630